SAP BusinessObjects

# Integrator's Guide

- SAP BusinessObjects Data Services 4.1 Support Package 1 (14.1.1.0)

2012-11-22

SAP

## Copyright



2012-11-22

# Contents

# Welcome to SAP BusinessObjects Data Services

## 1.1 Welcome

SAP BusinessObjects Data Services delivers a single enterprise-class solution for data integration, data quality, data profiling, and text data processing that allows you to integrate, transform, improve, and deliver trusted data to critical business processes. It provides one development UI, metadata repository, data connectivity layer, run-time environment, and management console—enabling IT organizations to lower total cost of ownership and accelerate time to value. With SAP BusinessObjects Data Services, IT organizations can maximize operational efficiency with a single solution to improve data quality and gain access to heterogeneous sources and applications.

## 1.2 Documentation set for SAP BusinessObjects Data Services

You should become familiar with all the pieces of documentation that relate to your SAP BusinessObjects Data Services product.

| Document | What this document provides |
|---|---|
| *Administrator's Guide* | Information about administrative tasks such as monitoring, lifecycle management, security, and so on. |
| *Customer Issues Fixed* | Information about customer issues fixed in this release. |
| *Designer Guide* | Information about how to use SAP BusinessObjects Data Services Designer. |
| *Documentation Map* | Information about available SAP BusinessObjects Data Services books, languages, and locations. |
| *Installation Guide for Windows* | Information about and procedures for installing SAP BusinessObjects Data Services in a Windows environment. |
| *Installation Guide for UNIX* | Information about and procedures for installing SAP BusinessObjects Data Services in a UNIX environment. |
| *Integrator's Guide* | Information for third-party developers to access SAP BusinessObjects Data Services functionality using web services and APIs. |

| Document | What this document provides |
|---|---|
| *Master Guide* | Information about the application, its components and scenarios for planning and designing your system landscape. Information about SAP BusinessObjects Information Steward is also provided in this guide. |
| *Management Console Guide* | Information about how to use SAP BusinessObjects Data Services Administrator and SAP BusinessObjects Data Services Metadata Reports. |
| *Performance Optimization Guide* | Information about how to improve the performance of SAP BusinessObjects Data Services. |
| *Reference Guide* | Detailed reference material for SAP BusinessObjects Data Services Designer. |
| *Release Notes* | Important information you need before installing and deploying this version of SAP BusinessObjects Data Services. |
| *Technical Manuals* | A compiled “master” PDF of core SAP BusinessObjects Data Services books containing a searchable master table of contents and index:<br>• *Administrator's Guide*<br>• *Designer Guide*<br>• *Reference Guide*<br>• *Management Console Guide*<br>• *Performance Optimization Guide*<br>• *Supplement for J.D. Edwards*<br>• *Supplement for Oracle Applications*<br>• *Supplement for PeopleSoft*<br>• *Supplement for Salesforce.com*<br>• *Supplement for Siebel*<br>• *Supplement for SAP*<br>• *Workbench Guide* |
| *Text Data Processing Extraction Customization Guide* | Information about building dictionaries and extraction rules to create your own extraction patterns to use with Text Data Processing transforms. |
| *Text Data Processing Language Reference Guide* | Information about the linguistic analysis and extraction processing features that the Text Data Processing component provides, as well as a reference section for each language supported. |
| *Tutorial* | A step-by-step introduction to using SAP BusinessObjects Data Services. |
| *Upgrade Guide* | Release-specific product behavior changes from earlier versions of SAP BusinessObjects Data Services to the latest release. This manual also contains information about how to migrate from SAP BusinessObjects Data Quality Management to SAP BusinessObjects Data Services. |
| *What's New* | Highlights of new key features in this SAP BusinessObjects Data Services release. This document is not updated for support package or patch releases. |

| Document | What this document provides |
|---|---|
| *Workbench Guide* | Provides users with information about how to use the Workbench to migrate data and database schema information between different database systems. |

In addition, you may need to refer to several Supplemental Guides.

| Document | What this document provides |
|---|---|
| *Supplement for J.D. Edwards* | Information about interfaces between SAP BusinessObjects Data Services and J.D. Edwards World and J.D. Edwards OneWorld. |
| *Supplement for Oracle Applications* | Information about the interface between SAP BusinessObjects Data Services and Oracle Applications. |
| *Supplement for PeopleSoft* | Information about interfaces between SAP BusinessObjects Data Services and PeopleSoft. |
| *Supplement for Salesforce.com* | Information about how to install, configure, and use the SAP BusinessObjects Data Services Salesforce.com Adapter Interface. |
| *Supplement for SAP* | Information about interfaces between SAP BusinessObjects Data Services, SAP Applications, SAP Master Data Services, and SAP NetWeaver BW. |
| *Supplement for Siebel* | Information about the interface between SAP BusinessObjects Data Services and Siebel. |

We also include these manuals for information about SAP BusinessObjects Information platform services.

| Document | What this document provides |
|---|---|
| *Information Platform Services Administrator's Guide* | Information for administrators who are responsible for configuring, managing, and maintaining an Information platform services installation. |
| *Information Platform Services Installation Guide for UNIX* | Installation procedures for SAP BusinessObjects Information platform services on a UNIX environment. |
| *Information Platform Services Installation Guide for Windows* | Installation procedures for SAP BusinessObjects Information platform services on a Windows environment. |

## 1.3 Accessing documentation

You can access the complete documentation set for SAP BusinessObjects Data Services in several places.

### 1.3.1 Accessing documentation on Windows

After you install SAP BusinessObjects Data Services, you can access the documentation from the Start menu.

1. Choose **Start > Programs > SAP BusinessObjects Data Services 4.1 > Data Services Documentation > All Guides**.
2. Click the appropriate shortcut for the document that you want to view.

### 1.3.2 Accessing documentation on UNIX

After you install SAP BusinessObjects Data Services, you can access the documentation by going to the directory where the printable PDF files were installed.

1. Go to `<LINK_DIR>/doc/book/en/`.
2. Using Adobe Reader, open the PDF file of the document that you want to view.

### 1.3.3 Accessing documentation from the Web

You can access the complete documentation set for SAP BusinessObjects Data Services from the SAP BusinessObjects Business Users Support site.

To do this, go to http://help.sap.com/bods.

You can view the PDFs online or save them to your computer.

## 1.4 SAP BusinessObjects information resources

A global network of SAP BusinessObjects technology experts provides customer support, education, and consulting to ensure maximum information management benefit to your business.

Useful addresses at a glance:

| Address | Content |
| --- | --- |
| Customer Support, Consulting, and Education services<br>http://service.sap.com/ | Information about SAP Business User Support programs, as well as links to technical articles, downloads, and online forums. Consulting services can provide you with information about how SAP BusinessObjects can help maximize your information management investment. Education services can provide information about training options and modules. From traditional classroom learning to targeted e-learning seminars, SAP BusinessObjects can offer a training package to suit your learning needs and preferred learning style. |
| Product documentation<br>http://help.sap.com/bods/ | SAP BusinessObjects product documentation. |
| Supported Platforms (Product Availability Matrix)<br>https://service.sap.com/PAM | Get information about supported platforms for SAP BusinessObjects Data Services.<br>Use the search function to search for Data Services. Click the link for the version of Data Services you are searching for. |

# Web service support

This section discusses both how an administrator can configure SAP BusinessObjects Data Services through the Administrator to publish jobs as callable web services, and how an application developer can access those web services.

The software publishes web services from the Management ConsoleAdministrator. To use SAP BusinessObjects Data Services as a web service, select the **Web Services** node in the Administrator's navigation tree. For general information on using the Administrator, see the *Management Console Guide*.

## 2.1 Overview

Web services are modular business applications based on open standards (WSDL, SOAP, and XML Schema primarily) that allow integration among different applications and environments through the Internet. Web services allow parts of existing applications to be used by other applications.

For business intelligence (BI), you can use web services to accomplish the following:

- Access legacy systems
- Conduct computer-to-computer interaction over an internal or external web
- Allow applications constructed in different languages on different platforms to communicate with each other in an enterprise environment

SAP BusinessObjects Data Services can:

- Publish any job as a callable web service (server functionality)
- Call published web services from within its jobs using the built-in web services datastore (client functionality)

If you have an application that also supports web services, you can use that application to run batch and real-time jobs or to publish your application's functionality to be called by data flows.

After you install the software, you can immediately start working with its client functionality because the built-in web services datastore is a web services client that provides access to a web services server from a data flow.

### Related Topics

- Using SAP BusinessObjects Data Services as a web service provider
- Consuming external web services in SAP BusinessObjects Data Services

## 2.2 Web services technologies

SAP BusinessObjects Data Services web services are fully compliant with Web Services Interoperability (WS-I) Basic Profile 1.0, and support three Java Web Services technologies.

| Web service technology | Description |
|---|---|
| SOAP | Connection protocol (envelope for XML messages) |
| WSDL | Language used to request a service and return replies (subset of XML) |
| XML Schema | Format used for the WSDL file |

The software supports SOAP 1.1, WSDL version 1.1 or 2.0, and Apache Axis 1.1 (an industry-standard SOAP message handler and WSDL parser).

SAP BusinessObjects Data Services is also compliant with the Microsoft .NET environment for web services. You can import the WSDL that the software generates into Visual Studio .NET and the web services datastore can call the WSDL that Visual Studio .Net generates.

**Note:**
The software does not support the Representational State Transfer (REST) web services architecture or the JSON message format.

### 2.2.1 SOAP

SAP BusinessObjects Data Services allows you to invoke real-time services using the following:

- Message Client API (which supports C++ and Java connections)
- TCP/IP
- proprietary XML using HTTP

In addition, the software supports the Simple Object Access Protocol (SOAP). SOAP is the industry standard from the World Wide Web Consortium (WC3.org) used to invoke network resources using XML over HTTP, HTTPS, and other standard protocols.

A SOAP gateway is built in to the Administrator. The software supports SOAP over HTTP and HTTPS protocols.

### 2.2.2 WSDL

Web Services Description Language (WSDL) is a subset of XML. It is used as a transport mechanism for XML messages. SAP BusinessObjects Data Services publishes its jobs in WSDL based on configuration settings applied in the Administrator, and then developers can create a web services client based on the software's WSDL.

The software also publishes all comments entered into the Designer's **Job Descriptions** box with each job that is added to the WSDL file.

The WSDL file generated by the software includes tags (such as services, bindings, ports, and operations) that support the use of the SOAP protocol. Each tag uses a name that the software provides. For example:

- You select which jobs to publish in the web service named `DataServices_server`. In WSDL, a service is a set of business operations with connection endpoints.
- Binding names include `Connection_Operations`, `Batch_Jobs`, `Real-time_Services`, and `Batch_Job_Admin`. WSDL uses bindings to associate operations with ports.
- Operation names have a one-to-one relationship with the names of batch jobs or real-time services.

### 2.2.3 XML Schema

WSDL uses XML Schemas to define input and output message formats.

- For server functionality, if a real-time service was defined with DTDs, you will need to translate the DTD format into the XML Schema format.
- For client functionality, the web services datastore imports metadata into SAP BusinessObjects Data Services using the XML Schema format only.

XML Schema formats are defined in the **types** element of the WSDL file.

**Note:**
When you import an XML schema for a real-time web service job, you should use a unique target namespace for the schema. When Data Services generates the WSDL file for a real-time job with a source or target schema that has no target namespace, it adds an automatically generated target namespace to the types section of the XML schema. This can reduce performance because Data Services must suppress the namespace information from the web service request during processing, and then reattach the proper namespace information before returning the response to the client.

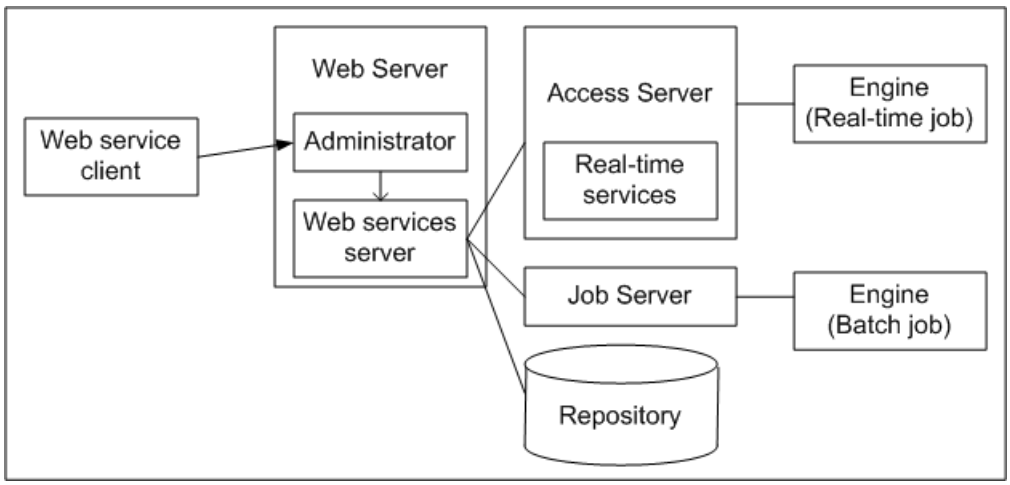
### 2.2.4 UDDI

UDDI is a method of publishing comments and other reference information about jobs to an external web site. SAP BusinessObjects Data Services does not publish information to a UDDI web site because most web service users work behind enterprise firewalls.

# Using SAP BusinessObjects Data Services as a web service provider

After the Administrator publishes batch or real-time jobs as web services, the web application hosts those web services. When an external application calls into SAP BusinessObjects Data Services through web services, the application acts as a web services client accessing a web services server.

Web service clients call the published web services, pass in the appropriate parameters, and receive the results. The software routes calls to the appropriate Job Server and job for processing.




Web services might be used in the following example scenarios:

- Dynamically update an internal web site

  Suppose you have an internal web site that manages foreign exchange rate status worldwide for the Finance department. When foreign exchange rates change more than a certain percentage, a batch job updates exchange rates in your financial data mart. The rate change initiates a call to a web service that starts the appropriate batch ETL job.

- Solve a processing issue

  Suppose you have an existing Enterprise Application Integration (EAI) bus infrastructure and want to manage batch processes and EAI transactional processes from within the same infrastructure. The transactional processes are complex. Their staging is laid out in the order process. However, EAI work flows do not have the ability to run batch processes.

The software can publish Web services that allow you to leverage EAI process management category tools (for example, webMethods Business Process Manager) to control and stage batch processes alongside its transactional processes.

The work flows might call the software to:

- Perform an initial load of a data mart for real-time reporting
- Refresh the data cache depending on specified business criteria
- Perform complex transforms on hierarchical objects for mapping data between ERP systems

## 3.1 WSDL basics

WSDL is a subset of XML that you can use to describe network services as a collection of endpoints capable of exchanging messages.

This table shows the elements in a WSDL file, and describes how those elements are used in the SAP BusinessObjects Data Services-generated WSDL file.

| Element Name | Description |
|---|---|
| definition | Root element |
| service | Used to group a set of related ports or endpoints to which a client application will connect. The software publishes a single service in the WSDL file it generates. |
| port | Defines a specific web service endpoint that a client can access. Each port has a unique name and a specific address used for binding. The software defines ports for web services as:<br>• `Connection_Operations`: used for authentication and ping<br>• `Real-time_Services`: used for real-time jobs exposed as web services<br>• `Batch_Jobs`: used for batch jobs exposed as web services (each batch jobs has its own operation)<br>• `Batch_Job_Admin`: used for administrative operations for batch jobs |
| portType | Defines a set of operations that a web service publishes.<br>A portType is bound to a particular port. The binding specifies the protocol and data formation for the operations defined by a portType. |

| Element Name | Description |
|---|---|
| operation | Defines a specific function call. The software publishes each batch job and real-time service as an operation. It also publishes connection operations. |
| message | Defines the data to transmit. There is an input (request) message, which the web service receives from the client, and there is an output (response) message, which the web service sends back to the client. |
| type | Defines the data types used in messages sent to/from a web service. |

**Related Topics**

- SoapAction element

## 3.1.1 Building a WSDL file

Use the information in the WSDL file produced by SAP BusinessObjects Data Services to create an application that can access batch jobs and real-time services. Access the WSDL file by making web service client calls to it using a reference URL.

To view the WSDL file so that you can create your application, use the **Web Services** node of the Management ConsoleAdministrator, or open a browser window and search for:

http://*hostname*:*port*/DataServices/servlet/webservices?ver=2.0&wsdlxml

### 3.1.1.1 To configure web service information using the Administrator

1. Open the Administrator.
2. Log in with Administrator-level privileges. Users with Monitor-level privileges cannot configure web services.

   **Note:**
   If you enable security for the WSDL file, SAP BusinessObjects Data Services requires that web services clients use the user name and password of any user with Administrator-level privileges to access all published web services.

3. Add connections from Access Servers and repositories to view jobs in the Administrator.

4. If you plan to publish real-time jobs as web services, configure real-time jobs as real-time services.

   The software publishes the following as web services:

   - Real-time services enabled as web service operations in the Administrator
   - Batch jobs enabled as web service operations in the Administrator
   - Connection Operations
     - Ping - Used to ping Web services
     - Logon and Logout - Security operations that provide controlled access to Web service operations (if enabled).

5. In the Administrator's navigation tree, select **Web Services**.

   The "Web Services Status" page opens. This page lists Web service operations that are published in the WSDL. By default, only the Ping operation is automatically published.

6. Click the **Web Services Configuration** tab.

   Use the **Configuration** tab to open the "Web Services Configuration" page. Use this page to select jobs and real-time services to be published, enable/disable security for the WSDL file, and to enable/disable access to full batch job attributes.

7. From the pull-down menu, use **Add Real-time Service** or **Add Batch Job** to add jobs or services to the WSDL, and click **Apply**.

   On the "Add Real-time Service" page, real-time services are grouped by the Access Server for which the service is configured. To add a real-time service to the WSDL, select an Access Server or select **All**, select the check box in front of a real-time service name, and click **Add**.

   On the "Add Batch Job" page, jobs are grouped by the repository on which the job is stored. To add a job to the WSDL, select a repository or select **All**, select the check box in front of a job name, and click **Add**.

8. (Optional) On the "Web Services Configuration" page, select **Enable Session Security** and click **Apply** to enable security for the WSDL.

   Security for published operations is disabled by default.

   With security enabled, instead of making a single call to the Administrator to start a batch job or trigger a real-time service from an external application, clients must make at least three calls:

   - The first call logs in to the Administrator and gets a session ID.
   - The second call accesses a job or service using the session ID as an input parameter. Create a call for each job or service you want to access.
   - The final call logs out of the session.

9. (Optional) On the "Web Services Configuration" page, from the drop-down menu, select **Enable Job Attributes** to allow the input message for all the batch jobs you publish to include all options supported for submitting batch jobs from the Administrator. The following table lists elements added to the message:

| Element | Description |
|---|---|
| job_system_profile | System profile used to run the job. |

| Element | Description |
| --- | --- |
| sampling_rate | Monitor sample rate (# of seconds). |
| auditing | Enable auditing (true or false). |
| recovery | Enable recovery (true or false). |
| job_server | Job Server or Server Group. |
| trace | Trace option to be enabled. You must specify an option to enable tracing. This element can be repeated for as many trace options as you require.<br>The WSDL defines values for the trace option and includes the following (all options on the batch job submit page of the administrator):<br>• job_trace_all<br>• job_trace_row<br>• job_trace_session<br>• job_trace_workflow<br>• job_trace_dataflow<br>• job_trace_transform<br>• job_trace_usertransform<br>• job_trace_userfunction<br>• job_trace_abapquery<br>• job_trace_sqlfunctions<br>• job_trace_sqlreaders<br>• job_trace_sqlloaders<br>• job_trace_optimized_dataflow<br>• job_trace_table<br>• job_trace_script<br>• job_trace_ascomm<br>• job_trace_rfc_function<br>• job_trace_table_reader<br>• job_trace_idoc_file<br>• job_trace_adapter<br>• job_trace_communication<br>• job_trace_parallel_execution<br>• job_trace_audit |

| Element | Description |
|---|---|
| distribution_level | You can distribute the execution of a job or a part of a job across multiple Job Servers within a Server Group to better balance resource-intensive operations.<br>You can specify the following values on the distribution level option when you execute a job:<br>• Job: A job can execute on an available Job Server.<br>• Data flow: Each data flow within a job can execute on an available Job Server.<br>• Sub data flow: A resource-intensive operation (such as a sort, table comparison, or table lookup) within a data flow can execute on an available Job Server.<br>**Note:**<br>Casing and spacing are important for these values. |

10. Navigate back to the "Web Services Status" page, choose the WSDL version you want to create, and click **View WSDL**.

    A new browser window opens with the WSDL displayed. Use the information in this file to perform the following:

    - Confirm that the software updated the WSDL file with all jobs and services without error.
    - Create calls to the software.

      Use the information in the WSDL file to configure your application to access batch jobs and real-time services.

      To ensure that your application calls the latest version of the job, update the WSDL when the metadata imported into the software changes for a job or real-time service by removing then re-adding a job or service from the "Web Services Configuration" page.

11. After your web service clients are accessing jobs, you can monitor the status of web service operations on the server by viewing the data on the "Web Services Status" page.

| Column name | Description |
|---|---|
| Operation Name | Same as job or real-time service name. |

| Column name | Description |
|---|---|
| Web Services Port | For jobs, the port name is Batch_jobs.<br>For services, the port name is Real-Time_Service.<br>For built-in operations, the port name is Connection_Operations.<br>For administrative operations for batch jobs, the port name is Batch_Job_Admin. |
| Repository/ Access Server | For jobs, the repository name.<br>For services, the Access Server name. |
| Job Information | For jobs, a link to the Batch Job History page.<br>For services, a link to the Real-time Services History page. |
| Requests Processed | Number of client requests successfully processed. |
| Requests Failed | Number of client requests that failed somewhere between the time that the Web Server receives the request and the Job Server receives it. |
| Requests Pending | Number of requests in a queue for Job Server. |
| Jobs Failed | Number of requests that failed due to a problem with the Job Server. |

### 3.1.2 Tips for using the WSDL file

The WSDL file:

- Appears in the View WSDL window or any browser window by searching for the following URL:
  http://*hostname:port*/DataServices/servlet/webservices?ver=2.0&wsdlxml

**Note:**
To support previously-created datastores using a WSDL file with XML schema simple types, manually delete "ver=2.0&" from the default URL of Web Service and Apply to save as follows:

http://*hostname:port*/DataServices/servlet/webservices?wsdlxml

- Displays all real-time services and jobs enabled for web services in the Administrator.
- Only displays log on, log off, and session ID information when security is enabled.
- Displays XML Schema formats in the **types** element.

### 3.1.3 WSDL versions

From time to time, the WSDL version used by SAP BusinessObjects Data Services may change for a variety of reasons. For example, the syntax may change in order to operate more efficiently, or to add support for new technologies.

Although we typically maintain backward compatibility between WSDL versions, we recommend that you move to the newest WSDL version available in your Data Services installation. The latest version often includes improvements in web service execution, and older versions may be deprecated and no longer supported over time.

For more information about the changes between specific WSDL versions, see the *Upgrade Guide*.

#### Version history

In general, a WSDL version is deprecated when the version of Data Services in which it was introduced is no longer supported.

| WSDL Version | Introduced | Deprecated |
|---|---|---|
| 1.0 | Original WSDL release | n/a |
| 1.1 | Data Integrator 11.5.0.0 | n/a |
| 2.0 | Data Integrator 11.7.0.0 | n/a |
| 2.1 | Data Services 12.2.1 | n/a |

## 3.2 Creating a client to use web services

To use a published web service, you must know the URL of the target WSDL. The Administrator produces a WSDL file with this URL: `http://hostname:port/DataServices/servlet/webservices?ver=2.0&wsdlxml`

The batch or real-time jobs must have previously been exposed as web services.

This section discusses general steps for using a published SAP BusinessObjects Data Services web service. The tools you use to develop your web services client are your choice and the exact steps in using those tools vary, but these basic steps apply as a simple overview to all development projects for web services clients.

1. Import the software's WSDL into your development environment to create a web services client application.

   The incorporated web services appear in the hierarchy of your development environment.

2. Open the web service.

   Each available port for the web service is made visible in the IDE.

3. Write the code to call any of the jobs or services provided by the ports.
4. Run the project to execute the code.

   Executing the code initiates the web services job. A connection is made to the web services tier of the Access Server.

The Access Server then sends information to various job servers, which then executes the al_engine process to run the job, and results are sent back to the Web services client application.

### 3.2.1 Design choices

SAP BusinessObjects Data Services provides different ways that you can call jobs using web services, each with benefits and drawbacks:

- Individually published job-specific web services (Batch_Jobs and Real-time_Services ports)

  These web services have their schema published directly in the WSDL, and web service development tools can automatically create classes that serialize and deserialize the input and output XML messages. However, you need to create a separate operation for each published job.

- Generalized web services (Batch_Jobs_Admin and Realtime_Service_Admin ports)

  These web services take a job name as input, but do not directly expose the input schema of the job. This allows an application to dynamically call different jobs with one web service, but the schema must be known in advance or generated dynamically with another web service call.

## 3.3 Supported web service operations

SAP BusinessObjects Data Services creates a WSDL file with a single service definition. It is possible to create multiple service definitions in a WSDL, but many web service implementations do not support more than one service definition. To avoid that limitation, the software creates only one service.

Within the service definition, the software defines ports for:

- Connection_Operations
- Batch_Job_Admin
- Real-time_Services
- Batch_Jobs
- Repo_Operations

### 3.3.1 Connection port

SAP BusinessObjects Data Services generates WSDL that defines connection operations that belong to web services. The software supports the following Connection operations.

| Operation | Description |
|---|---|
| Ping | Verifies the connection to web services |
| Logon | Verifies secure access before establishing a session |
| Logout | Terminates a session |

**Note:**
The software generates Logon and Logout operations only if you enable security for published jobs.

#### 3.3.1.1 Ping

The Ping operation is an empty input message with a ping operation request. The output message is a text string that returns the current SAP BusinessObjects Data Services version, which indicates that a connection has been established.

#### 3.3.1.2 Logon

The Logon operation is required when you enable SAP BusinessObjects Data Services to provide secure communication. To access web services, provide an Administrator login name and password (with Administrator-level privileges). When the software validates them, the logon operation returns an Administrator session ID that you must include in all subsequent calls to the web services.

#### Input message

| Element | Type | Description |
|---|---|---|
| cms_authentication | String | Specifies the type of authentication to use for logging on to a Data Services web service. Values include:<br>• `secEnterprise`: Use for Enterprise authentication<br>• `secLDAP`: Use for LDAP authentication<br>• `secWinAD`: Use for Windows Active Directory authentication<br>• `secSAPR3`: Use for SAP authentication |
| cms_system | String | The CMS server name. If the CMS server is listening on the default 6400 port, then pass only the CMS server name. If the server is listening on any other port, then also pass the port number. |
| password | String | Password for the CMS user. |
| username | String | CMS user name. |

#### Output message

| Element | Type | Description |
|---|---|---|
| SessionId | String | A unique session ID is returned. Use this session ID in subsequent Data Services operations for which session security is enabled. |

#### 3.3.1.3 Logout

The Logout operation is required when you enable SAP BusinessObjects Data Services to provide secure communication. When web service communication is complete, call the Logout operation to terminate the session.

### 3.3.2 Realtime_Service_Admin port

#### 3.3.2.1 Get_RTMsg_Format

Use Get_RTMsg_Format to retrieve the input/output format for a real-time service as an XML Schema. The real-time service does not need to be published as a web service.

#### Input message

| Element | Type | Description |
|---|---|---|
| serviceName | string | The name of the real-time service as displayed in the Administrator. |
| selector | string | A selector that determines whether the input or output schema for the service is returned. Valid strings include:<br>• in - Returns the input schema.<br>• out - Returns the output schema. |

#### Output message

| Element | Type | Description |
|---|---|---|
| schema | string | The input or output XML Schema for the real-time service. |
| rootElement | string | The root element of the returned XML Schema. |
| rootElementNS | string | The root element namespace of the returned XML Schema. |
| schemaName | string | The name of a dependent schema used in the returned XML Schema, if applicable. This element may be returned multiple times. |
| schema | string | A dependent schema used in the returned XML Schema, if applicable. This element may be returned multiple times. |
| errorMessage | string | Any error message that resulted while retrieving the XML Schema for the real-time service. |

### 3.3.2.2 Get_RTService_List

Use the Get_RTService_List operation to retrieve a list of the names of published real-time services.

#### Input message

Get_RTService_List takes no input message.

#### Output message

| Element | Type | Description |
|---|---|---|
| serviceName | string | The list of published real-time services. |
| errorMessage | string | Any error message that occurred while retrieving the list of real-time services. |

### 3.3.2.3 Run_Realtime_Service

Use Run_Realtime_Service to call a published real-time service. The real-time service must be running and published as a web service in the Administrator, and the XML input content must match the input format defined for the real-time service.

#### Input message

| Element | Type | Description |
|---|---|---|
| serviceName | string | The name of the real-time service as displayed in the Administrator. |
| xmlInput | string | The XML input content used to start the real-time service. This content must match the input format required by the real-time service. |

#### Output message

| Element | Type | Description |
|---|---|---|
| xmlOutput | string | The XML output content returned by the real-time service. This content is formatted according to the output schema of the real-time service called. |
| errorMessage | string | Any error message that resulted while attempting to call the real-time service. |

## 3.3.3 Batch_Job_Admin port

#### 3.3.3.1 Get_BatchJob_List

Use the Get_BatchJob_List operation to retrieve a list of the names of published batch jobs.

**Input message**

| Element | Type | Description |
|---|---|---|
| repoName | string | The name of the repository to access. This parameter is optional. |
| allBatchJobs | boolean | Includes all batch jobs in the repository, not only those published as web services. This parameter is optional. |

**Output message**

| Element | Type | Description |
|---|---|---|
| jobName | string | The list of published batch jobs. This element has an additional attribute, repo, which specifies the name of the repository that contains the job. |
| errorMessage | string | Any error message that occurred while retrieving the list of batch jobs. |

#### 3.3.3.2 Get_BatchJob_RunIDs

Each individual run of an SAP BusinessObjects Data Services batch job is assigned a unique run ID.

Use the Get_BatchJob_RunIDs operation to retrieve a list of run IDs associated with a particular batch job.

#### Input message

| Element | Type | Description |
|---|---|---|
| jobName | string | The name of the batch job. |
| status | string | The status code for the type of run IDs requested. Valid codes include: `running`, `succeeded`, `error`, `warning`, and `all`. |
| repoName | string | The name of the repository to access. When specified, the operation returns only runIDs from this repository. This element is optional. |

#### Output message

The response of the Get_BatchJobs_RunIDs operation contains one or more run element. Each run element contains the following sub-elements:

| Element | Type | Description |
|---|---|---|
| runID | integer | The unique ID for the batch job run. |
| status | string | The status code for the batch job run. Valid codes include: `running`, `succeeded`, `error`, `warning`, and `all`. |
| repoName | string | The repository name associated with the batch job. |
| errorMessage | string | Any error message that occurred while retrieving the list of batch jobs. |

### 3.3.3.3 Get_BatchJob_Status

Use the Get_BatchJob_Status operation to retrieve the status of a particular batch job run.

#### Input message

| Element | Type | Description |
|---|---|---|
| runID | integer | The run ID for the particular batch job status desired. |
| repoName | string | The name of the repository to access. |

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | integer | The status for the operation. Valid values include:<br>• 0 - The operation completed successfully.<br>• 1 - The operation encountered an error. For example, the repoName specified is invalid. |
| status | string | The status of the batch job run. Valid values include:<br>• Running - The job is currently running.<br>• Succeeded - The job completed successfully with no errors.<br>• Warning - The job completed successfully but warnings occurred.<br>• Error - The job completed with an error. |

### 3.3.3.4 Get_Error_Log

SAP BusinessObjects Data Services produces several types of log information for a batch job published as a web service.

Use the Get_Error_Log operation to retrieve the error log for a batch job.

#### Input message

| Element | Type | Description |
|---|---|---|
| runID | integer | The batch job run ID for the particular log desired. |
| repoName | string | The name of the repository to access. |
| page | integer | The page number of the error log to return. This element is optional. |

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | integer | The status for the operation. Valid values include:<br>• 0 - The operation completed successfully.<br>• 1 - The operation failed to retrieve the error log. |
| error | string | The error log associated with the input batch job run ID. |

### 3.3.3.5 Get_Job_Input_Format

Use Get_Job_Input_Format to retrieve the input format for a batch job as an XML Schema.

#### Input message

| Element | Type | Description |
|---|---|---|
| jobName | string | The name of the batch job as displayed in the Administrator. |
| repoName | string | The name of the repository to access. |

#### Output message

| Element | Type | Description |
|---|---|---|
| format | string | The input format for the batch job, as an XML Schema. |
| errorMessage | string | Any error message that resulted while retrieving the input format for the batch job. |

### 3.3.3.6 Get_Monitor_Log

SAP BusinessObjects Data Services produces several types of log information for a batch job published as a web service.

Use the Get_Monitor_Log operation to retrieve the monitor log for a batch job.

#### Input message

| Element | Type | Description |
|---|---|---|
| runID | integer | The batch job run ID for the particular log desired. |
| repoName | string | The name of the repository to access. |
| page | integer | The page number of the monitor log to return. This element is optional. |

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | integer | The status for the operation. Valid values include:<br>• 0 - The operation completed successfully.<br>• 1 - The operation failed to retrieve the monitor log. |
| monitor | string | The monitor log associated with the input batch job run ID. |

### 3.3.3.7 Get_Trace_Log

SAP BusinessObjects Data Services produces several types of log information for a batch job published as a web service.

Use the Get_Trace_Log operation to retrieve the trace log for a batch job.

#### Input message

| Element | Type | Description |
|---|---|---|
| runID | integer | The batch job run ID for the particular log desired. |
| repoName | string | The name of the repository to access. |
| page | integer | The page number of the trace log to return. This element is optional. |

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | integer | The status for the operation. Valid values include:<br>• 0 - The operation completed successfully.<br>• 1 - The operation failed to retrieve the trace log. |
| trace | string | The trace log associated with the input batch job run ID. |

### 3.3.3.8 Run_Batch_Job

Use Run_Batch_Job to call a batch job with the ability to specify job parameters and global variables.

#### Input message

| Element | Type | Description |
|---|---|---|
| jobName | string | The name of the batch job as displayed in the Administrator. |
| repoName | string | The name of the repository to access. |
| jobServer | string | The name of the job server to use to execute the job. This element is optional, but cannot be specified if serverGroup is also specified. |
| serverGroup | string | The name of the server group to use to execute the job. This element is optional, but cannot be specified if jobServer is also specified. |
| jobParameters | complex | A complex XML element that sets specific job execution parameters. |
| globalVariables | complex | A complex XML element that defines global job execution variables. |

**Note:**

- If you do not specify a serverGroup or jobServer, the operation attempts to find an available job server that is attached to the repository by first checking the server group list, and then the job server list.

- For detailed information about the jobParameters and globalVariables elements, view the WSDL from the Administrator. For more information about available job parameters and global variables, see the *Reference Guide*.

**Output message**

| Element | Type | Description |
|---|---|---|
| pid | int | The process ID number for the batch job execution. Process IDs can be reused. |
| cid | int | The counter ID number for the batch job excecution. You can use a counter ID together with a process ID to uniquely identify a specific instance of the job execution. |
| rid | int | The run ID associated with the specific execution of the batch job. |
| repoName | string | The repository name associated with the batch job execution. |
| errorMessage | string | Any error message that resulted while attempting to call the batch job. |

### 3.3.3.9 Stop_Batch_Job

Use the Stop_Batch_Job operation to stop a particular batch job run.

#### Input message

| Element | Type | Description |
|---|---|---|
| runID | integer | The run ID for the particular batch job run to stop. |
| repoName | string | The name of the repository to access. |

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | integer | The success code for the batch job stop attempt. Valid codes include:<br>• 0 - The operation successfully stopped the specified batch job.<br>• 1 - The operation failed to stop the specified batch job in the specified repository. |
| errorMessage | string | The error message associated with a failure to stop a batch job run. |

### 3.3.4 Real-time_Services port

SAP BusinessObjects Data Services generates WSDL that defines how to invoke real-time services enabled as web service operations. Each real-time service name is represented as an operation name in the WSDL file.

Each real-time service operation has a set of messages that it uses to communicate with the real-time service. Real-time services use a defined XML message as input and a defined XML message as output. Real-time services obtain the format of these messages from the repository and provide the format in the WSDL.

The software supports XML Schemas as its message format for real-time services. A web service provides only XML Schemas in the WSDL. You will need to convert any DTDs to XML schemas as necessary.

The repository stores XML Schemas that define the input and output messages as independent definitions. The WSDL file includes these definitions in the **types** element.

The messages that an XML Schema defines for each real-time service operation are:

- Header message

  If security is enabled for the message, the software defines a secure session identifier in the message header.

- Input message

  When an external web services client invokes it, the input message passes information to a real-time service. The name of the input message is the name of the operation that the software publishes

followed by the suffix `_Input`. The input message contains the message source defined by the real-time service.

- Output message

  The software returns the output message when the real-time service completes. The output message contains the output generated by the real-time service. The name of the output message is the name of the operation followed by the suffix `_Output`. The output message contains the message target defined by the real-time service.

- Fault message

  The software returns a fault message when it cannot invoke the real-time service.

#### 3.3.4.1 Message formats

The following segment shows the syntax that Data Services generates in a WSDL file to define an operation's messages. In this example:

- `RTService` represents the name of the real-time service as defined in the Administrator.
- `XMLSchemaName` represents the name of the XML Schema that was used to create an XML message source or XML message target in the Designer.
- `RootElement` represents the root element of the XML Schema.

The software publishes a WSDL that includes input and output XML Schema message formats in the **types** element.

**Note:**
Server support for real-time services requires that you use a valid URL for locating XML Schema (.xsd) files in an import statement. A local file name cannot be used. For example, the .xsd must be either self-contained when imported into the software or it must use a network reference (URL), not a file or relative path reference, as an identifier.

The WSDL file displays the operations for real-time services within a `portType` tag.

### 3.3.5 Batch_Jobs port

SAP BusinessObjects Data Services generates WSDL that defines how to start batch jobs. The WSDL file represents each batch job name as an operation.

In addition, the WSDL file defines an input and output message for each operation. An input message communicates the input needed by the job at startup (such as the global variables needed to start the job). An output message either confirms that the job started or presents the errors that prevent the job from starting.

WSDL defines the following messages for each operation:

- Header message

  When security is enabled for the message, the software defines a secure session identifier in the message header.

- Input message

  The input message passes information needed by the batch job at startup. The name assigned to the input message is the name of the operation followed by the suffix `_Input`. The input message contains global variables.

  When security is enabled for the message, the software defines a secure session identifier in the message header.

- Output message

  The software returns the output message when the batch job starts. The output message contains the job identification. The name of the output message is the name of the operation followed by the suffix `_Output`. The output message contains the following IDs:

  - OS process ID of the started job
  - Job Server Counter ID of the started job

- Fault message

  The software returns a fault message if the batch job fails to start. It returns a text description of the error that prevents the job from starting.

#### 3.3.5.1 SoapAction element

The definition of each batch job operation uses the `soapAction` element to define the batch job name needed to launch the job.

The WSDL file displays the `soapAction` element in the service and port section.

#### 3.3.5.2 Security

When publishing a job as a web service, the Administrator can enable secure access, requiring that web services clients provide authentication and authorization (an Administrator username and password) for access to the web service operations. Administrator-level (not Monitor-level) privileges must be used. That is, you cannot limit access to users based on role. This authentication is SSL-compliant.

If you do not enable secure access and you are using HTTP, web services clients have open access to all published batch jobs and real-time services.

**Related Topics**

- To configure web service information using the Administrator

## 3.3.6 Repo_Operations port

SAP BusinessObjects Data Services generates WSDL that defines operations that belong to web services. The software supports the following operations on the Repo_Operations port.

| Operation | Description |
|---|---|
| Delete_Repo_Objects | Deletes objects from the repository. |
| Import_Repo_Object | Imports an object to the repository. |
| Validate_Repo_Object | Validates an object contained in the repository. |
| Export_DQReport | Exports reports to a specified location at runtime. |

### 3.3.6.1 Delete_Repo_Objects

Use the Delete_Repo_Objects operation to delete objects from the SAP BusinessObjects Data Services repository.

### Input message

| Element | Type | Description |
|---|---|---|
| objName | string | The name of the object to delete from the repository. This element requires the attribute objType and can occur multiple times.<br>The objType attribute specifies the type of the object:<br>• BATCH_JOB<br>• REALTIME_JOB<br>• WORKFLOW<br>• DATAFLOW<br>• ABAP_DATAFLOW<br>• DATA_QUALITY_TRANSFORM_CONFIGURATION<br>• DATASTORE<br>• FILE_FORMAT<br>• XML_SCHEMA<br>• DTD<br>• CUSTOM_FUNCTION<br>• EXCEL_WORKBOOK<br>• COBOL_COPYBOOK<br>• SYSTEM_PROFILE<br>• SUBSITUTION_CONFIGURATION<br>• PROJECT<br>• TABLE<br>• TEMPLATE_TABLE<br>• DOMAIN<br>• HIERARCHY<br>• STORED_PROCEDURE<br>• IDOC<br>• BW_MASTER_TRANSFER_STRUCTURES<br>• BW_MASTER_TEXT_TRANSFER_STRUCTURES<br>• BW_TRANSACTION_TRANSFER_STRUCTURES<br>• BW_HIERARCHY_TRANSFER |
| repoName | string | The name of the repository that contains the objects to delete. |
| jobServer | string | The name of the job server associated with the repository. This element is optional, but cannot be specified if serverGroup is also specified. |
| serverGroup | string | The name of the server group associated with the repository. This element is optional, but cannot be specified if jobServer is also specified. |
| traceOn | string | Enables tracing for the operation. This element is optional. |

**Note:**

If you do not specify a serverGroup or jobServer, the operation attempts to find an available job server that is attached to the repository by first checking the server group list, and then the job server list.

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | int | The status of the operation:<br>• 0 - The operation completed successfully.<br>• 1 - The operation failed to complete successfully. |
| errorMessage | string | The error message associated with the operation. This element is output only if the operation fails to complete successfully. |
| traceMessage | string | The trace message associated with the operation. This element is output only if the traceOn element is specified on input. |

### 3.3.6.2 Import_Repo_Object

Use the Import_Repo_Object operation to save an XML object definition to the SAP BusinessObjects Data Services repository.

#### Input message

| Element | Type | Description |
|---|---|---|
| definition | string | The object to import to the repository. The object must be defined in XML format. |
| repoName | string | The name of the repository in which to import the object. |
| jobServer | string | The name of the job server associated with the repository. This element is optional, but cannot be specified if serverGroup is also specified. |
| serverGroup | string | The name of the server group associated with the repository. This element is optional, but cannot be specified if jobServer is also specified. |
| traceOn | string | Enables tracing for the operation. This element is optional. |
| passphrase | string | The passphrase that was used to encode any passwords in the XML object definition.<br>**Note:**<br>If the passphrase you specify is incorrect, the operation will still save the XML objects to the repository, but any passwords in the definition will be removed. |

**Note:**
If you do not specify a serverGroup or jobServer, the operation attempts to find an available job server that is attached to the repository by first checking the server group list, and then the job server list.

#### Output message

| Element | Type | Description |
|---|---|---|
| returnCode | int | The status of the operation:<br>• 0 - The operation completed successfully.<br>• 1 - The operation failed to complete successfully. |
| errorMessage | string | The error message associated with the operation. This element is output only if the operation fails to complete successfully. |
| traceMessage | string | The trace message associated with the operation. This element is output only if the traceOn element is specified on input. |

### 3.3.6.3 Validate_Repo_Object

Use the Validate_Repo_Object operation to validate an object stored in the SAP BusinessObjects Data Services repository.

### Input message

| Element | Type | Description |
|---|---|---|
| objName | string | The name of the object to validate. |
| objType | string | The type of the object to validate:<br>• BATCH_JOB<br>• REALTIME_JOB<br>• WORKFLOW<br>• DATAFLOW<br>• ABAP_DATAFLOW<br>• DATA_QUALITY_TRANSFORM_CONFIGURATION<br>• CUSTOM_FUNCTION |
| repoName | string | The name of the repository that contains the object to validate. |
| systemProfile | string | The name of the job system profile to use while validating the object. This element is optional. |
| jobServer | string | The name of the job server associated with the repository. This element is optional, but cannot be specified if serverGroup is also specified. |
| serverGroup | string | The name of the server group associated with the repository. This element is optional, but cannot be specified if jobServer is also specified. |
| substitutionParameters | complex | Substitution parameters to override while validating the object. This element is optional and contains one or more parameter child elements. |
| parameter | string | An individual substitution parameter. |
| traceOn | string | Enables tracing for the operation. This element is optional. |

**Note:**
If you do not specify a serverGroup or jobServer, the operation attempts to find an available job server that is attached to the repository by first checking the server group list, and then the job server list.

**Output message**

| Element | Type | Description |
|---|---|---|
| returnCode | int | The status of the operation:<br>• 0 - The operation completed successfully.<br>• 1 - The operation failed to complete successfully. |
| errorMessage | string | The error message associated with the operation. This element is output only if the operation fails to complete successfully. |
| traceMessage | string | The trace message associated with the operation. This element is output only if the traceOn element is specified on input. |

### 3.3.6.4 Export_DQReport

Use the Export_DQReport operation to export reports to a specified location at runtime.

### Input message

| Element | Type | Description |
|---|---|---|
| runID | integer | The unique ID for the batch job run. |
| repoName | string | The name of the repository to that contains the batch job. |

### Output message

| Element | Type | Description |
|---|---|---|
| exportFileName | string | The name of the file that is exported; for example, `matchcriteriasummary_Set1.pdf`. |
| exportPath | string | The path where the reports will be exported to. The default path is `<DS_COMMON_DIR>\DataQuality\reports\`. Upon execution, the repository name and job name folders are appended to the path. If the Overwrite option is not selected, a run ID folder is also appended to the path.<br>**Note:**<br>If you export reports to a location other than a local drive, such as a network drive, before you execute the job you must start the web application server with an account that has access rights to that location. |
| exportStatus | boolean | The status of the overall export operation:<br>• 0 - The operation completed successfully.<br>• Negative integer - The operation failed to complete successfully. |
| processMessage | string | Informational messages about the overall export process. |
| reportName | string | The name of the report that is displayed in the Management Console; for example, Match Criteria Summary. |
| reportStatus | integer | The status of the export operation for each report:<br>• 0 - The operation completed successfully.<br>• Negative integer - The operation failed to complete successfully. |
| statusMessage | string | Informational message about the export status for each report. |

## 3.4 Optimizing real-time web service performance

You can modify the connection pool settings for SAP BusinessObjects Data Services' real-time web services. You can optimize the performance of your installation by configuring the connection pool to

match your Access Server and hardware configuration. Connection pool configuration settings are found in the <*DS_COMMON_DIR*>/conf/admin.xml file.

| Setting | Description |
|---|---|
| ws-conn-max-active | Controls the maximum number of connections that can be borrowed from the pool at one time. When the value is exceeded, the pool is exhausted. Negative values allow unlimited connections. The default value for this setting is 8. |
| ws-conn-max-idle | Controls the maximum number of connections that can sit idle in the connection pool at any time. Negative values allow unlimited idle connections. The default value for this setting is 8. |
| ws-conn-min-idle | Controls the minimum number of connections that can sit idle in the connection pool at any time. The default value for this setting is 0. |
| ws-conn-when-exhausted-action | Specifies the action to perform when the connection pool is exhausted. Possible values include:<br>• fail<br>Throws an exception.<br>• grow<br>Creates and returns a new connection. This can exceed the maximum specified in ws-conn-max-active.<br>• block<br>Blocks requests until a new or idle connection is available. |
| ws-conn-max-wait | Specifies, in milliseconds, how long to block requests when ws-conn-when-exhausted-action is set to block. Negative values block requests indefinitely. |

After you modify admin.xml, restart your web application server to activate the new settings.

## 3.5 Enabling SSL support

### 3.5.1 To configure SSL on the web application server

For SAP BusinessObjects Data Services web services to work with SSL, the web application server must be configured to support SSL connections. The server.xml file can be used to configure the packaged Tomcat application server.

**Note:**
For other web application servers, refer to the product documentation about how to configure SSL support.

1. Open `server.xml` in a text editor. This file is located in the `Tomcat55\conf` directory at the same level as `LINK_DIR`.
2. Locate the commented `connector` element in the XML:

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443 -->
<!--
<Connector port="8443" maxHttpHeaderSize="8192"
  maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
  enableLookups="false" disableUploadTimeout="true"
  acceptCount="100" scheme="https" secure="true"
  clientAuth="false" sslProtocol="TLS" />
 -->
```

3. Remove the comment (`<!-- -->`) tags around the `connector` element.
4. Add the `keystoreFile` and `keystorePath` attributes into the `connector` element.

```
keystoreFile="path/to/keystore/file"
keystorePass="keystore_password"
```

5. Restart the Tomcat application server.

## 3.6 Error reporting

SAP BusinessObjects Data Services uses web services to define every operation with both an input and output message. In addition to the output message, the software returns a fault message when an error occurs.

### 3.6.1 Administrator log

In addition to the fault message, SAP BusinessObjects Data Services writes log and debug messages to the Administrator's log file (`webadmin.log`). Fault messages include a short description of a failure. For detailed information about an error, use the Administrator's log file.

All Administrator components share the Administrator's log. The software prefixes these messages with the name of the component that issues the error message. For web services, the component name is the name of the Java class issuing the error. All web service classes start with `com.acta.adapter.webservice`.

The software creates the Administrator's log file in: `<DS_COMMON_DIR>\log\webadmin.log`.

To control the level of detail in the `webadmin.log` file, you must edit the `log4j.properties` file. The properties file is located in:

`<LINK_DIR>\ext\webserver\webapps\acta_web_admin\WEB-INF`

To obtain a debug trace of events, change the log level from the default of `INFO` to `DEBUG`. For example, `log4j.rootLogger=DEBUG, A`

### 3.6.2 Web service log

In addition to the shared Administrator log, web service messages are also written to a separate log file. The `WebService.log` file is in `<DS_COMMON_DIR>\log`.

### 3.6.3 Error messages

The following are error messages that you might encounter if you are using SAP BusinessObjects Data Services as a web service provider to accept inbound calls:

- A web service is unable to process a request due to an unknown function in the soapAction element.

  The server returns this error message if the soapAction header in the HTTP request is not recognized. Every web service call expects a soapAction header that indicates an action. The WSDL publishes a soapAction for each operation. When the web services server cannot determine what action to take, it is unable to call the software.

  To find extended error information, use the `WebService.log` file in `<DS_COMMON_DIR>/log`. To use extended diagnostics, use debug tracing in the `webadmin.log` file.

- A web service is unable to process a request to call real-time service *ServiceName* using Access Server *AccessServerName*.

  The server returns this error message when it recognizes a request to call a real-time service but is unable to extract the XML message from the SOAP Envelope that is supposed to be sent to the real-time service. To find extended error information, use the `WebService.log` file in `<DS_COMMON_DIR>/log`.

  To use extended diagnostics, use debug tracing in the `webadmin.log` file.

- A web service sent a request to invoke real-time service *ServiceName* to Access Server *AccessServerName*. The request failed with error = *<access-server generated error message>*

  The server returns this error message if it recognizes a request to call a real-time service, data was extracted from the incoming SOAP Envelope, and data was passed to the Access Server, which refused to service the request.

To locate where the error occurred use the Access Server log file (**Real-time > *AccessServerName* > Logs - Current**).

If the Access Server passed the request on to the Job Server, use the following logs to diagnose the problem:

- Job Server log (<*DS_COMMON_DIR*>/log/*JobServerName*/server_eventlog.txt)
- Real-time service provider log (**Real-time > *AccessServerName* > *Real-timeServiceName* > *ProcessID***)

- A web service operation is unable to process the request to start batch job *JobName* on server *JobServerName*. Error = *<web services generated error message>*

  The server returns this error message if it recognizes a request to start a batch job but is unable get the information it needs to start the job.

  To find extended error information, use the WebService.log file in <*DS_COMMON_DIR*>/log. To use extended diagnostics, use debug tracing in the webadmin.log file.

- A web service sent a request to start batch job *JobName* on server *JobServerName*. The Job Server refused the request with error: *<job server generated error message>*

  The server returns this error message if it recognizes a request to start a batch job and has passed the request to the Job Server to start the job. The Job Server is unable to start the job. To find extended error information, use the Job Server log (<*DS_COMMON_DIR*>/log/*JobServer Name*/server_eventlog.txt).

### Related Topics

- Administrator log

# Consuming external web services in SAP BusinessObjects Data Services

You can add functionality to SAP BusinessObjects Data Services to invoke web services in external applications from data flows. This functionality requires configuring the software's built-in web services datastore type. The web services datastore provides support for locating and importing metadata for a web services server as well as invoking web service operations.

The web services datastore works by sending a request and waiting until it receives a reply from a web services server.

For example, you might create a web services server as a front-end to a legacy application. You could call the web services server daily from a data flow to access inventory and update an inventory data mart.

The interaction between the web services datastore and an external web service has these parts:

- Creating a web services datastore that identifies the WSDL, which describes the web services server.
- Importing metadata to extract the information form the WSDL needed to access the web service server.
- Creating a data flow that uses the imported function call to call the web services server.

## 4.1 To access a web service using the Designer

To configure access to a specific web services, use the Designer to create a web service datastore. SAP BusinessObjects Data Services provides access to web services as stream-oriented function calls, which it configures when you import metadata.

When you configure a web service datastore, specify the URL of the web services server for a data flow to access. It must be the same URL that accepts a web service connection and returns the WSDL.

The datastore connects to the web services server using the URL to locate the definition of published services.

1. Create a web service datastore.

| Parameter | Details |
|---|---|
| **Datastore type** | Choose Web Service. |
| **Web Service URL** | Specify the location of the external web service to accept a connection and return WSDL. |
| **User name** | Enter the user name for HTTP basic authentication. Required only if basic authentication is needed to connect to the web service provider. |
| **Password** | Enter the password for HTTP basic authentication. Required only if basic authentication is needed to connect to the web service provider. |
| **XML Recursion Level** | Enter the number of passes the software should run through the XSD to resolve names. The default is 0. |
| **Keystore path** | If the web service provider uses an SSL connection, specify the location of the keystore used to establish the connection. |
| **Socket timeout** | Enter the maximum number of milliseconds the web service client will wait to receive the response from the web service provider. |
| **Axis2/c config file path** | Enter the path to your Axis2/c configuration file (`axis2.xml`). If a path is not specified, the default path is `<LINK_DIR>/ext/webservice-c/axis2.xml`. |
| **Proxy host** | Enter the host name for the HTTP proxy server. Required only if a proxy server is needed to connect to the web service provider. |
| **Proxy port** | Enter the port number for the HTTP proxy server. Required only if a proxy server is needed to connect to the web service provider. |
| **Proxy username** | Enter the user name for the HTTP proxy server. Required only if a proxy server that uses authentication is needed to connect to the web service provider. |
| **Proxy password** | Enter the password for the HTTP proxy server. Required only if a proxy server that uses authentication is needed to connect to the web service provider. |

2. Import metadata from the web service datastore
   a. From the object library, open the web service datastore.

      The Designer calls the web service server at the indicated WSDL URL and obtains a list of the published services and ports.

   b. Expand the ports to see published operations available for import.
   c. Right-click an operation and select **Import**.

      The software imports web service operations as function calls and lists them under the web service datastore in the object library. Each function call includes a definition for both the input and output messages required for communication with a web service operation. The Designer extracts the details about the request and reply messages and generates XML Schema that describes the messages.

3. From the Designer, add a web service function call to a job.

As a web services client, the software calls a web services server twice:

- During design time to import metadata for the functions and data types that a particular web service supports.
- During run time to call the web service and invoke its functionality.

For more information, see “Defining a web service datastore” in the *Designer Guide*.

## 4.2 To add web service calls to a job

Once a web service datastore is created and metadata is imported, you can add web service function calls to an SAP BusinessObjects Data Services job.

1. Add a Query transform to the data flow.
2. Open the Query editor, right-click the target schema and select **New function call**.

   The Function Editor opens listing the operation metadata that you imported under the datastore name.
3. Select a datastore to view the metadata that you want to add to your job.
4. Select the metadata name and click **Next**.
5. Map the input schema to the output schema.

   **Note:**
   If you want to nest data in the target schema, use this first Query transform to place the schema in your job and additional Query transforms to perform the nesting. The Function Editor does not allow complex schema configuration.
6. Click **OK**.

   The imported schema appears in the query.
7. Configure the rest of the data flow by supplying input to the function call and extracting the response information obtained from the web service.

## 4.3 Configuring HTTP header fields

In the HyperText Transfer Protocol (HTTP), header fields generally contain the operating parameters of an HTTP request or response. The header fields define various characteristics of the the data transfer that is requested or the data that is provided in the message body.

An HTTP header field always starts with a field name, then a colon (`:`), and ends with the field value. A core set of header fields is standardized by the Internet Engineering Task force. These standard headers are commonly understood by all compliant protocol implementations. Header field names may also be any application-specific strings, known as customary headers.

### 4.3.1 To configure standard HTTP header fields

Standard HTTP header fields have specified values that are the same and fixed for all web service functions in the web service datastore. The values for standard fields also remain the same for all web service calls in a dataflow.

1. In the Designer, open the web service datastore configuration and click **Advanced**.
2. Select **Standard HTTP header fields** and click **...**.
   The "Edit HTTP Header Fields" screen appears.
3. Enter each field name and its corresponding default value and click **OK**.
   A semicolon-separated list of the header fields appears in the column for **Standard HTTP header fields**.
4. Click **OK** to accept the datastore configuration.

### 4.3.2 To configure dynamic HTTP header fields

Dynamic HTTP header fields have values that may be different for each function in the web service datastore. The values for dynamic fields can also change for each web service call in a dataflow.

1. In the Designer, open the web service datastore configuration and click **Advanced**.
2. Select **Dynamic HTTP header fields** and click **...**.
   The "Edit HTTP Header Fields" screen appears.
3. Enter each field name and the maximum length for its value and click **OK**.
   A semicolon-separated list of the header fields appears in the column for **Dynamic HTTP header fields**.
4. Click **OK** to accept the datastore configuration.

When you import a function into the web service datastore, the dynamic HTTP headers are available for mapping in the HTTPHeader schema.

## 4.4 Enabling SSL support

### 4.4.1 To configure SSL on the native web service datastore

To configure SSL support on the native web service datastore, add the path to your keystore to the datastore configuration.

**Note:**
The keystore path is only used while importing WSDL operations into the datastore, and is not used at runtime.

For more information about configuring web service datastores, see the *Designer Guide*.

### 4.4.2 To configure SSL in the runtime execution file

1. Obtain a certification authority (CA) certificate for the client.
2. Open `LINK_DIR\ext\webservice-c\axis2.xml` in a text editor.
3. Locate the commented `transportReciever` and `transportSender` elements in the XML:

```
<transportReceiver name="https" class="axis2_http_receiver">
  <parameter name="port" locked="false">6060</parameter>
  <parameter name="exposeHeaders" locked="true">false</parameter>
</transportReceiver>

<transportSender name="https" class="axis2_http_sender">
  <parameter name="PROTOCOL" locked="false">HTTP/1.1</parameter>
</transportSender>
```

4. Remove the comment (`<!-- -->`) tags around the `transportReciever` and `transportSender` elements.
5. Provide the path to the CA certificate as the `SERVER_CERT` parameter.

```
<parameter name="SERVER_CERT">/path/to/ca/certificate</parameter>
```

6. If you need client authentication, additionally provide the private key and keystore passphrase.

```
<parameter name="KEY_FILE">/path/to/client/certificate/chain/file</parameter>
<parameter name="SSL_PASSPHRASE">passphrase</parameter>
```

## 4.5 Enabling WS-Security support

WS-Security is a communications protocol that applies security to web services at the message level. The protocol defines how integrity and confidentiality can be enforced on web service messaging, as well as how to attach signatures and encryption headers to SOAP messages. In addition, it defines how to attach security tokens such as X.509 certificates or Kerberos tickets to messages.

SAP BusinessObjects Data Services implements WS-Security support through the Apache open source project rampart/c and can be configured through the web service datastore and an external policy file.

### 4.5.1 To configure WS-Security on the native web service datastore

1. Create a security policy file (`policy.xml`) based on the WS-Security policy specification that satisfies your security requirements.

   You can see sample policy files by downloading a copy of rampart/c and opening the `samples/secpolicy` folder.
2. Place your security policy file in the `LINK_DIR/ext/webservice-c` folder or another location.
3. If your policy file is not named `policy.xml` or is located in a folder other than the default location, specify the path using the WSS Policy file path parameter in the datastore configuration.
4. Enable the rampart/c module in your `axis2.xml` file.

   Within `axis2.xml`, uncomment the `<!-- module ref="rampart" /-->` line. By default, `axis2.xml` is installed to `LINK_DIR/ext/webservice-c/`.

You can also make more changes in the Advanced section of the native web service datastore configuration:

| Parameter | Details |
|---|---|
| WSS Username | Enter the username to use for WS-Security. |
| WSS Password | Enter the password to use for WS-Security. |
| WSS Password type | Enter the password type to use for WS-Security. The available options are PlainText and Digest. |
| WSS Time to live | Enter the time for WS-Security protected messages to live. The default is 0. Any positive number will add a timestamp to the message. |
| WSS Policy file path | Enter the path to your WS-Security policy file. The default path is `LINK_DIR/ext/webservice-c/policy.xml`. |

For more information about configuring web service datastores, see the *Designer Guide*.

**Related Topics**

- WS-Security policy specification

# Using the Message Client API

You can integrate SAP BusinessObjects Data Services' real-time services by using the C++ or Java API. Either of these interfaces allows you to connect to the real-time service with a persistent connection to the server, send and receive data from it, and close the connection.

**Note:**
The Message Client API supports the creation of reports, similar to any job you run with the software.

In the execution of real-time jobs with real-time services APIs, these steps take place:

1. An administrator logs into the Management Console and chooses which real-time jobs to expose as real-time services. Those job names are stored in the local repository.
2. An administrator chooses which Access Server to run the services on and starts the real-time services.
3. A developer accesses a real-time service through Java and C++ libraries.
4. A C++ or Java application client makes a connection to the Access Server, which then sends information to various job servers.
5. The job servers route requests to an engine to process the real-time job.

### File location

The Message Client API files for each supported platform are installed to *LINK_DIR*\SDK\RTSDK. When the software is installed on a Windows server, the Message Client API files for both C++ and Java for each UNIX platform are provided in a `.tar.gz` archive.

To use the UNIX Message Client API files with a Windows installation, copy the appropriate Message Client API package file for your UNIX platform from `LINK_DIR\SDK\RTSDK\platform` to your UNIX system, and then unzip and extract the archive to the desired installation location. For example, on Solaris:

```
gunzip MessageClient_Solaris_64bit.tar.gz
tar -xvf MessageClient_Solaris_64bit.tar
```

## 5.1 Interface components

The interface between the Access Server and your application includes these components:

- Connection definition (Connection)

A class that defines the connection that your application uses to send and receive messages from the Access Server. Initialize the class (using the connect method) each time you initialize your application.

- Connection initialization (Connect)

  A method that creates the connection using host and port information supplied by the client.

- Request (Invoke)

  A method that indicates the request message for the specified real-time service. This method is a synchronous call that waits for a return.

- Exception handlers (Error message)

  A class that returns exceptions thrown by the connection object and system exceptions, if available.

## 5.2 Creating the connection

The Connection object creates an active connection to the Access Server.

Creating a Connection (calling the Connect method) does the following:

- Authenticates the client as secure
- Produces an open TCP/IP socket between the client and the Access Server
- Encapsulates the connection information into a client identifier (Connection ID)

As soon as you create the Connection object, you can use it to send messages to the Access Server. Typically, you would create a single Connection per client. If you attempt to call the Connect method for a Connection that already exists, the Access Server ignores the call.

## 5.3 Sending messages

Send requests from the client application using the Invoke method and the Connection ID.

Each business operation implemented by your web application can result in a call to the Access Server with a message. The Access Server uses the name of the business operation to determine the path for the message. When you use SAP BusinessObjects Data Services to process real time jobs, you pair this business operation name, called a service, with the job and data flow names you defined in the software to process the message. There is a one-to-one correlation between business operation, service, job, and XML source.

Call the Invoke method with a string return value to process a synchronous response.

## 5.4 Closing the connection

The library provides a method (Disconnect) with the Connection object that allows you to systematically close the TCP/IP socket between the client and the Access Server.

## 5.5 Pseudo code example

```
// Login and authenticate the client connection = connect(  accessServerAddress,
// TCP/HTTP address  clientName,
// matches Access Server  clientPassword);
// IP & Client
// security settings
// Invoke Service String xmlOut = connection.invoke(  serviceName,
// has mapping to RT job  xmlIn);
// according to the RT job DTD
// In case of an error returns the error code
// and error message
```

## 5.6 C++ API reference

### 5.6.1 Class RTServiceClient

#### RTServiceClient

Contains C++ methods for allowing a client to connect to real-time services.

| Method summary | |
|---|---|
| `virtual void` | connect (char* *`hostname`* unsigned short *`port`* bool *`use_SSL`* char* *`trusted_certs_filename`*) |
| `virtual char*` | invoke (char* serviceName char* inData ) |
| `virtual void` | disconnect ( ) |

#### Constructor detail

#### RTServiceClient

```
RTServiceClient(){}
```

#### Method detail

#### connect

```
virtual void connect(char* hostname, unsigned short port)
```

Establishes a connection between a client and the Access Server. You must establish a connection before a message can be exchanged.

`hostname` – the name or IP address of the machine that hosts the Access server.

`port` – the port number used for the connection.

#### setUseSSL

```
void setUseSSL(const char* certificate_dir)
```

Uses SSL when communicating with the Access Server.

`certificate_dir` – the folder that contains the certificate files. Files with `.crt`, `.pem`, and `.cer` extensions are used as trusted certificates.

#### invoke

```
virtual char* invoke(char* serviceName, char* inData)
```

Sends the input data to the real-time service and returns the output data.

`serviceName` – the name of the real-time service to invoke.

`inData` – the input data to send to the real-time service.

#### disconnect

```
virtual void disconnect ()
```

Stops the connection between a client and the Access Server.

## 5.6.2 Class RTServiceClientError

#### RTServiceClientError

Represents an error object thrown by the C++ class RTServiceClient.

| Method summary |
| --- |
| RTServiceClientError(const char*, int=0) |
| RTServiceClientError(const char*, const char*, const char*) |
| RTServiceClientError(const RTServiceClientError&) |

#### Method detail

#### RTServiceClientError

```
RTServiceClientError(const char*, int=0)
```

```
RTServiceClientError(const char*, const char*, const char*)
```

```
RTServiceClientError(const RTServiceClientError&)
```

Represents an error object thrown by the client library.

## 5.7 Java API reference

### 5.7.1 Class RTServiceClient

#### com.businessobjects.rtsclient.RTServiceClient

Contains Java methods for allowing a client a connection to real-time services.

| Method Summary | |
| --- | --- |
| `public void` | connect ( char * *`machineName`* int *`port`* bool *`use_SSL`* char* *`trusted_certs_filename`* ) throws RTServiceException-Throws: |
| `public java.lang.String` | invoke (java.lang.String *`serviceName`* java.lang.String *`inData`* ) throws RTServiceExceptionThrows: |
| `public void` | disconnect ( ) throws RTServiceExceptionThrows: |

#### Method detail

#### connect

```
public void connect (java.lang.String machineName, int port, bool use_SSL, char* trusted_certs_filename)
throws RTServiceExceptionThrows:
```

Establishes a connection between a client and the Access Server. You must establish a connection before a message can be exchanged.

`machineName` – the name or IP address of the machine that hosts the Access Server.

`port` – the port number used for the connection.

`use_SSL` – indicates that SSL should be used for the connection to the Access Server.

`trusted_certs_filename` – the path to the file that contains the trusted SSL certificates.

### invoke

```
public java.lang.String invoke(java.lang.String serviceName, java.lang.String inData) throws RTServiceExcep
tionThrows:
```

Sends the input data to the real-time service and returns the output data.

`serviceName` – the name of the real time service to invoke.

`inData` – the input data to send to the real time service.

### disconnect

```
public void disconnect ()throws RTServiceExceptionThrows:
```

Stops the connection between a client and SAP BusinessObjects Data Services.

# Using the JMS adapter

## 6.1 Introduction

### 6.1.1 About this section

This section provides a detailed step-by-step method of installing and configuring the SAP BusinessObjects Data Services JMS adapter. It includes a description of required support software, including supported versions, details of the adapter components, environment setup both for the software and external applications, and instructions for executing the adapter.

#### 6.1.1.1 Who should read this section?

This section assumes the following:

- You understand how to use Designer to design and run real-time data flows (RTDFs) and batch jobs.
- You have a basic understanding of how to use Administrator to administer SAP BusinessObjects Data Services processes. (You administer adapters from the Administrator.)
- You have a working knowledge of the environment this adapter is targeting.
- You know the role an adapter plays in business systems integration.
- You have some familiarity with XML and XML configuration schemas.
- Also, to integrate the software with an external system, it's recommended that you be familiar with systems administration and systems integration issues.

General SAP BusinessObjects Data Services product documentation assumes the following:

- You are an application developer, consultant or database administrator working on data extraction, data warehousing, or data integration.
- You understand your source and target data systems, DBMS, legacy systems, business intelligence, and messaging concepts.
- You understand your organization's data needs.

- If you are interested in using this product to design real-time processing you are familiar with:
  - DTD and XML Schema formats for XML files
  - Publishing Web Services (WSDL, HTTP/S and SOAP protocols, etc.)
- You are familiar with the software's installation environments: Microsoft Windows or UNIX.

### 6.1.2 Adapter overview

Typical enterprise infrastructure is a complex mix of off-the-shelf and custom applications, databases, ERP applications etc. SAP BusinessObjects Data Services combines and extends critical Extraction Transformation Loading (ETL) and Enterprise Application Integration (EAI) technology components required for true enterprise data integration.

Integrating disparate applications with the software's platform requires adapters. These adapters help facilitate otherwise incompatible applications and systems work together, thereby sharing data.

#### 6.1.2.1 About Java Messaging Service (JMS)

Enterprise-messaging or Message Oriented Middleware (MOM) products are fast becoming an essential component for integrating intra-company operations. They allow separate business components to be combined into a reliable, yet flexible, system. In addition to the traditional MOM vendors, several database vendors and Internet-related companies also provide enterprise-messaging products.

Java language clients and Java language middle-tier services must be capable of using these messaging systems. Java Messaging Service (JMS) provides a common way for Java language programs to access these systems.

JMS is a set of interfaces and associated semantics that define how a JMS client accesses the facilities of an enterprise-messaging product. Since messaging is peer-to-peer, all users of JMS are generically referred to as clients. A JMS application is composed of a set of application-defined messages and a set of clients that exchange them. Products that implement JMS do this by supplying a provider that implements the JMS interfaces.

#### 6.1.2.2 Scope of the JMS adapter

- SAP BusinessObjects Data Services initiates Request/Reply

  The software initiates the request by sending the message on a pre-configured request queue and gets the reply on a pre-configured reply queue.

- The software initiates Request/Acknowledgment

  The software initiates the request by sending the message on a pre-configured target queue or by publishing a message to a JMS topic. In this case, only the acknowledgment is sent back to the software.

- IR initiates Request/Acknowledgment & Request/Reply

  In this case, an external Information Resource (IR is a JMS compatible application) sends the requests to the software and gets the reply or acknowledgment.

  Alternatively, the IR publishes a message to a JMS topic to which the JMS adapter has subscribed.

## 6.2 Installation and configuration

### 6.2.1 JMS adapter installation

This section details the components of the Adapter for JMS as well as system requirements.

The Adapter for JMS is automatically installed when you install SAP BusinessObjects Data Services version 12.0.0 or later.

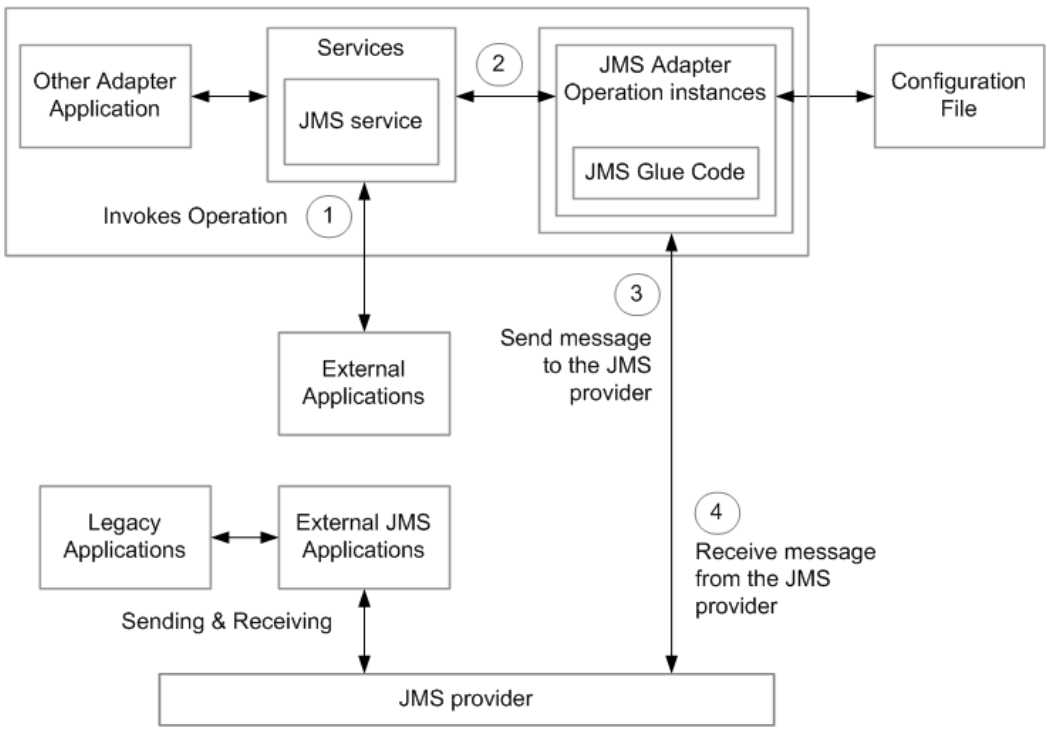
#### 6.2.1.1 System prerequisites

Before you install your SAP BusinessObjects Data Services Adapter for JMS, ensure that the following software is installed. For specific installation instructions, see the technical documentation for each product.

| Software | Version | Comment |
|---|---|---|
| JMS Provider | | For example, Weblogic Application Server |
| SAP BusinessObjects Data Services | 11.7.0 or later | Use the software to configure the services and adapter |
| SAP BusinessObjects Data Services Adapter SDK | 2.0.0.0 or later | |

### 6.2.1.2 Adapter product components

The following diagram shows a functional overview of the SAP BusinessObjects Data Services Adapter for JMS with other components and their potential interrelationships:




The diagram shows the architecture and functionality of the SAP BusinessObjects Data Services Adapter for JMS as well as how the adapter interacts with the external JMS application through the JMS Provider. The adapter sends or receives data on queues using the Point to Point (P2P) mode of communication, or publishes or subscribes to a JMS topic using the Publish/Subscribe mode of communication.

The flow of control in the previous diagram is as follows:

1. External application invokes a service on the software.
2. Based on the service invoked on the software, its respective real-time data flow (RTDF) invokes the Operation instance with XML data sent by the external application as input.
3. This operation instance sends the message to the configured queue or topic in the JMS provider. Based on the type of operation (such as Request/Reply or Request/Acknowledge), the JMS provider sends the Reply/Acknowledgment message back to the software.
4. External JMS application sends messages to the JMS Provider on a request queue or publishes the message to a topic. The JMS Adapter receives these messages after polling them from the JMS Provider and for P2P, sends the reply back to external JMS application on a configured reply queue. No reply is sent if the message was from a topic.

### 6.2.2 JMS adapter configuration

Before the Adapter for JMS can begin integrating the JMS Provider with the SAP BusinessObjects Data Services system you must create and configure at least one adapter instance and at least one operation for each instance. Adapter instances identify the JMS Application used in the integration. Adapter operations identify the integration operations to be used with the configured adapter instance.

Operations provided with Adapter for JMS include the following:

- PutGet Operation (Request/Reply): The software initiates a request, sending a message on a pre-configured request queue. Simultaneously, the software listens on a pre-configured reply queue. An external JMS-compatible application listens on the request queue, processes the request, and returns an XML response message to the reply queue. The adapter sends the message to the Job service.
- Put Operation (Request/Acknowledgment): The software initiates a request, sending a message on a pre-configured target queue. If the message was sent successfully, the adapter sends an acknowledgement to the Job service. The adapter raises an exception if it was unable to send the message.
- Get Operation (Request/Acknowledgment and Request/Reply from Information Resource): An external information resource (IR) sends a request XML message to a JMS queue. The adapter polls the JMS queue at a time interval you specify in the configuration. When the adapter receives a message from the JMS queue, it sends the message to the pre-configured Job service.

  After processing the XML message, the Job service may send a response message to the adapter. When this happens, the adapter puts the message in a pre-configured response queue. If the response queue is not configured, it becomes a request/acknowledgment operation and no reply is sent. If there is any error in invoking another service from the Job service, the original message is sent to the undelivered queue for reference by the IR.
- PutTopic Operation (Request/Acknowledgment): A software service initiates a request, publishing an XML message to a pre-configured target topic. If the message was sent successfully, the adapter sends an acknowledgement to the Job service. The adapter raises an exception if it was unable to send the message.
- GetTopic Operation (Request/Acknowledgment): An external information resource (IR) publishes an XML message to a JMS topic. The adapter polls the topic at the time intervals specified in the

configuration. When the adapter receives the message from the topic, it sends the message to the service that handles the message.

### 6.2.2.1 To configure the JMS adapter

All SAP BusinessObjects Data Services adapters communicate with the software through a designated Adapter Manager Job Server. Install adapters on the computer containing your designated Adapter Manager Job Server. This special Job Server integrates adapters with the software using the Administrator and Designer. After you install your adapter:

1. Use the Server Manager utility to configure adapter connections with the Adapter Manager Job Server.
2. From the Administrator, perform the following tasks:
   - Add at least one instance of the adapter to system.
   - Add at least one operation for each adapter instance.
   - Start the adapter instance (operations are started automatically).
3. Open the Designer and create an adapter datastore. Use metadata accessed through the adapter to create batch and/or real-time jobs.

For more information, see “To configure Job Servers” in the *Installation Guide* and “Adapter Considerations” in the *Management Console Guide*.

### 6.2.2.2 To configure an adapter instance in the Administrator

From the Administrator you can add a JMS adapter to the SAP BusinessObjects Data Services system as well as edit existing adapter configurations. Add the adapter in the Administrator before you run jobs that use information from that adapter.

1. Select **Adapter Instances** > **Job Server**.
2. Select the **Configuration** tab.
3. Click **Add**.
4. Select **JMSAdapter** from the list of adapters available on this Job Server and click **Apply**.
5. Enter the required information to create a JMS Adapter instance and click **Apply**.

The Administrator makes the adapter instance available to the software.

#### 6.2.2.2.1 Adapter instance configuration information

To configure a JMS adapter instance in SAP BusinessObjects Data Services, you need to complete the fields in the Administrator under Adapter instance startup configuration.

| Field | Description |
|---|---|
| **Adapter Instance Name** | Enter a unique name that identifies this instance of the adapter. |
| **Access Server Host** | Enter the host ID of the computer running the Access Server that connects to this adapter instance. To run a real-time job, you must configure a service that the Access Server will use to run the job. When a job uses adapter-based data, the Access Server must be able to connect to the adapter instance. |
| **Access Server Port** | The message broker port of the Access Server host. After you log into the Administrator for this Access Server, select **Configuration > Client Interfaces** to view message broker port information. |
| **Adapter Retry Count** | Applies if adapter instance fails or crashes. Enter 0 for no retries; enter a negative number for indefinite retries. |
| **Adapter Retry Interval** | Wait in msec. between adapter retry attempts. |
| **Classpath** | The adapter is a Java program, so you must configure the jar files required by the adapter `CLASSPATH`. The adapter is pre-configured with all necessary jar files except for the vendor-specific JMS provider jar files. Add your JMS provider's jar files to the `CLASSPATH`. For example:<br>• `<LINK_DIR>/lib/acta_adapter_sdk.jar`<br>• `<LINK_DIR>/lib/acta_broker_client.jar`<br>• `<LINK_DIR>/lib/acta_tool.jar`<br>• `<LINK_DIR>/ext/lib/xerces.jar`<br>• `<LINK_DIR>/lib/acta_jms_adapter.jar`<br>• `<LINK_DIR>/ext/lib/jms/<JMS Provider Jar File>`<br>**Note:**<br>Specify the jar file provided with the JMS provider that you are using. For Weblogic, the name of jar file is `weblogic.jar`. |
| **Autostart** | When set to True, the adapter interface automatically starts when the Administrator starts. |
| **Trace Mode** | Set this flag to control the number of trace messages the adapter produces. There are two settings:<br>• True: The adapter interface writes information and error messages to help debug problems. The adapter writes information and error messages to the *adapter_instance_name*`_trace.txt` file in the `<`*DS_COMMON_DIR*`>\adapters\logs` directory.<br>• False: The adapter interface writes only error information messages. The adapter writes error messages to the *adapter_instance_name*`_error.txt` file in the `<`*DS_COMMON_DIR*`>\adapters\logs` directory. |
| **Additional Java Launcher Options** | Additional command line parameters used for the `javaw.exe` command line and for the adapter itself. (See specific adapter documentation for details.) |

| Field | Description |
|---|---|
| **Adapter Type Name** | (Read-only) Name of the adapter used to create this instance. |
| **Adapter Version** | (Read-only) Version of the adapter used to create this instance. |
| **Adapter Class** | (Read-only) Name that identifies the adapter class. The name depends on the type of adapter. |

In the JMS Adapter section, select a Configuration Type and enter Configuration parameters.

| Parameter | Description |
|---|---|
| **Configuration Type** | Uses only the configuration parameters associated with the selected configuration type.<br>• JNDI configuration type<br>• MQ configuration type |

For the JNDI configuration type, use the following configuration parameters.

| Parameter | Description |
|---|---|
| **Server URL** | Represents the URL of the JMS Provider. For example: `t3://<JMS Provider IP Address>:<port number>`. |
| **JNDI Context Factory** | JNDI context factory name is JMS Provider specific. You can choose the context factory from a list that includes common context factories. If you require a context factory that is not listed, you can add it to the list by editing file `<DS_COMMON_DIR>/adapters/config/templates/JMSAdapter.xml` and updating the `<jndiFactory>` element. For Weblogic as a JMS Provider, the JNDI Factory name is: `weblogic.jndi.WLInitialContextFactory`. |
| **Queue Connection Factory** | Queue connection factory name. For example: `JMSConnections.AdapterConnectionFactory`. |
| **Topic Connection Factory** | Topic connection factory name. For example: `JMSConnections.AdapterTopicConnectionFactory`. |

For the MQ configuration type, use the following configuration parameters.

| Parameter | Description |
|---|---|
| **MQ Queue Manager Name** | (Optional) Specify if not using the default MQ Queue Manager on the system running MQ. |
| **MQ Channel Name** | (Optional) Specify if not using the default MQ Channel on the system running the adapter. |
| **MQ Computer Name** | (Optional) Specify if not using the MQ Queue Manager on the same system running the adapter. |
| **MQ Port** | (Optional) Specify if not using the default MQ port (1414). |
| **MQ User ID** | (Optional) Specify if required to login to the MQ Queue Manager. |
| **MQ Password** | (Optional) Specify if required to login to the MQ Queue Manager. |

### 6.2.2.3 To add an operation instance to an adapter instance

1. Select **Configuration** > **Adapter instances**.
2. Click **Operations** under Dependent Objects.
3. Click **Add** to configure a new operation. Or, you can click the link of an existing operation instance to edit its configuration.
4. Select an operation type from the list and click **Apply**. The options that appear on this page vary based on operation-specific design.

Complete the operation instance configuration form and click **Apply**.

#### 6.2.2.3.1 Operation instance configuration options

Each operation type contains different configuration options. Operations include:

- Put Operation (request/acknowledgment) options
- PutTopic Operation (request/acknowledgment) options
- PutGet Operation (request/reply) options
- Get Operation (request/reply and request/acknowledgment) options
- GetTopic Operation (request/acknowledgment only) options

**Note:**
When specifying a queue or topic, you must provide the JNDI queue name or the MQ queue name as indicated by the Adapter Configuration Type property.

##### *Put Operation (request/acknowledgement) options*

To set up an operation instance of type Put Operation in SAP BusinessObjects Data Services, complete the following fields in the Administrator.

| Field | Description |
|---|---|
| **Operation instance** | The unique operation instance name. In the Designer, your operation metadata object is imported with this name. |
| **Thread count** | The number of copies of Request/Reply operation to run in parallel. For parallel (asynchronous) processing of messages coming from real-time service, more than one copy should be used. But if the sequence of messages is important (synchronous processing), more than one thread should not be used. (Multiple copies of real-time services must be supported by multiple instances of Request/Reply.) The default is 1. |
| **Operation retry count** | The number of times to retry this operation if it fails. Enter 0 to indicate no retries are to be attempted. Enter a negative number to indicate the operation should be retried indefinitely. |
| **Operation retry interval** | The time (in milliseconds) to wait between operation retry attempts. |
| **Display name** | The display name of the operation instance. This display name is visible in the Designer's metadata browsing window. |
| **Description** | The description of the operation instance. This description is visible in the Designer's metadata browsing window. |
| **Enable** | Whether to enable the operation to start at the same time as the adapter instance. Valid values are true and false.<br>• When true, the operation starts when the adapter instance starts.<br>• When false, the operation needs to be started manually from "Adapter Operations Status" window of the adapter administrator. |
| **Destination Queue** | The name of the destination queue where the message will be sent. |
| **Request Format** | The DTD or XSD file name that defines the XML message used in the operation. |
| **Request XML Root Element** | The name of the XML root element. |

### *PutTopic Operation (request/acknowledgement) options*

To set up an operation instance of type PutTopic in the SAP BusinessObjects Data Services, complete the following fields in the Administrator.

| Field | Description |
|---|---|
| **Operation instance** | The unique operation instance name. In the Designer, your operation metadata object is imported with this name. |
| **Thread count** | The number of copies of Request/Reply operation to run in parallel. For parallel (asynchronous) processing of messages coming from real-time service, more than one copy should be used. But if the sequence of messages is important (synchronous processing), more than one thread should not be used. (Multiple copies of real-time services must be supported by multiple instances of Request/Reply.) The default is 1. |
| **Operation retry count** | The number of times to retry this operation if it fails. Enter 0 to indicate no retries are to be attempted. Enter a negative number to indicate the operation should be retried indefinitely. |
| **Operation retry interval** | The time (in milliseconds) to wait between operation retry attempts. |
| **Display name** | The display name of the operation instance. This display name is visible in the Designer's metadata browsing window. |
| **Description** | The description of the operation instance. This description is visible in the Designer's metadata browsing window. |
| **Enable** | Whether to enable the operation to start at the same time as the adapter instance. Valid values are true and false.<br>• When true, the operation starts when the adapter instance starts.<br>• When false, the operation needs to be started manually from "Adapter Operations Status" window of the adapter administrator. |
| **Destination Topic** | The topic to which the operation is published. Use JNDI or MQ name as specified by Adapter Configuration Type. |
| **Message Format** | The DTD or XSD file name defining the XML message used in this operation. |
| **Request XML Root Element** | The name of the XML root element. |
| **Persistent Message** | Whether to make published messages available to durable subscribers. Valid values are true and false. When true, published messages are available to durable subscribers. |

### *PutGet Operation (request/reply) options*

To set up an operation instance of type PutGet Operation in SAP BusinessObjects Data Services, complete the following fields in the Administrator.

| Field | Description |
|---|---|
| **Operation instance** | The unique operation instance name. In the Designer, your operation metadata object is imported with this name. |
| **Thread count** | The number of copies of Request/Reply operation to run in parallel. For parallel (asynchronous) processing of messages coming from real-time service, more than one copy is used. If the sequence of messages is important (synchronous processing), more than one thread should not be used. (Multiple copies of real-time services must be supported by multiple instances of Request/Reply.) The default is 1. |
| **Operation retry count** | The number of times to retry this operation if it fails. Enter 0 to indicate no retries are to be attempted. Enter a negative number to indicate the operation should be retried indefinitely. |
| **Operation retry interval** | The amount of time (in milliseconds) to wait between operation retry attempts. |
| **Display name** | The display name of the operation instance. This display name is visible in the Designer's metadata browsing window. |
| **Description** | The description of the operation instance. This description is visible in the Designer's metadata browsing window. |
| **Enable** | Whether to enable the operation to start at the same time as the adapter instance. Valid values are true and false.<br>• When true, the operation starts when the adapter instance starts.<br>• When false, the operation needs to be started manually from "Adapter Operations Status" window of the adapter administrator. |
| **Request Queue** | The name of the destination queue where the message will be sent. |
| **Reply Queue** | The name of the destination queue where the message will be sent. |
| **Timeout** | The maximum time (in milliseconds) the operation should wait for the reply message. |
| **Continue After Error** | Whether to continue after encountering an error. Valid values are true and false.<br>• When true, the operation instance remains in start stage even after the error.<br>• When false, the operation instance stops after the error occurs during the process. |
| **Request Format** | The DTD or XSD file name that defines the Request XML message used in this operation. |

| Field | Description |
|---|---|
| **Request XML Root Element** | The name of the XML root element in the Request DTD or XSD. |
| **Reply Format** | The DTD or XSD file name that defines the Reply XML message used in the operation. |
| **Reply XML Root Element** | The name of the XML root element in the Reply DTD or XSD. |

### *Get Operation (request/reply and request/acknowledgement) options*

To set up an operation instance of type Get Operation in SAP BusinessObjects Data Services, complete the following fields in the Administrator.

| Field | Description |
|---|---|
| **Operation instance** | The unique operation instance name. In the Designer, your operation metadata object is imported with this name. |
| **Polling interval** | The time interval (in milliseconds) for polling the source queue by this operation instance. For example, If the polling interval is 1000, then it polls the source queue after every one second. |
| **Operation retry count** | The number of times to retry this operation if it fails. Enter 0 to indicate no retries are to be attempted. Enter a negative number to indicate the operation should be retried indefinitely. |
| **Operation retry interval** | The time (in milliseconds) to wait between operation retry attempts. |
| **Enable** | Whether to enable the operation to start at the same time as the adapter instance. Valid values are true and false.<br>• When true, the operation starts when the adapter instance starts.<br>• When false, the operation needs to be started manually from "Adapter Operations Status" window of the adapter administrator. |
| **Source Queue** | The name of the queue where the message is sent by the IR and received by the adapter. Use JNDI or MQ name as specified by the Adapter Configuration Type. |
| **Service** | The name of the real-time service invoked by the operation when it receives a new message from the Source Queue. |
| **Timeout** | The maximum time (in milliseconds) that the Service takes to process a message. If the operation instance is unable to invoke the service within the Timeout limit, it sends the error message to the undelivered queue. |

| Field | Description |
|---|---|
| **Continue After Error** | Whether to continue after encountering an error. Valid values are true and false.<br>• When true, the operation instance remains in start stage even after the error.<br>• When false, the operation instance stops after the error occurs during the process. |
| **Default Response Queue** | [optional]: Used only for Request/Reply operation. In case of Request/Acknowledgment operation, it remains blank. The application sends the reply back to external JMS application (IR) on this queue. Use JNDI or MQ name as specified by the Adapter Configuration Type. |
| **Undelivered Queue** | [optional]: The undelivered queue for receiving the error messages, if any. Use JNDI or MQ name as specified by the Adapter Configuration Type. |
| **Request DTD Root Element** | The name of the root element for the input DTD for this operation. |

### *GetTopic Operation (request/acknowledgement only) options*

To set up an operation instance of type GetTopic in SAP BusinessObjects Data Services, complete the following fields in the Administrator.

| Field | Description |
|---|---|
| **Operation instance** | The unique operation instance name. In the Designer, your operation metadata object is imported with this name. |
| **Polling interval** | The time interval (in milliseconds) for polling the source topic by this operation instance. For example, if the polling interval is 1000, then it polls the source topic after every one second. |
| **Operation retry count** | The number of times to retry this operation if it fails. Enter 0 to indicate no retries are to be attempted. Enter a negative number to indicate the operation should be retried indefinitely. |
| **Operation retry interval** | The time (in milliseconds) to wait between operation retry attempts. |
| **Enable** | Whether to enable the operation to start at the same time as the adapter instance. Valid values are true and false.<br>• When true, the operation starts when the adapter instance starts.<br>• When false, the operation needs to be started manually from "Adapter Operations Status" window of the adapter administrator. |
| **Source Topic** | The topic to which the operation subscribes. Use JNDI or MQ name as specified by Adapter Configuration Type. |

| Field | Description |
|---|---|
| **Durable subscriber** | The subscription name of Durable subscriber. If not applicable, leave this field blank. |
| **Service** | The name of the real-time service invoked by the operation when it receives a new message from the source topic. |
| **Timeout** | The maximum time (in milliseconds) that the service takes to process a message. |
| **Continue After Error** | Whether to continue after encountering an error. Valid values are true and false.<br>• When true, the operation instance remains in start stage even after the error.<br>• When false, the operation instance stops after the error occurs during the process. |

### 6.2.2.4 Defining a JMS adapter datastore

Use the SAP BusinessObjects Data Services Adapter for JMS with a batch job or real-time data flow (RTDF) when the batch job or RTDF passes a message to an operation instance, using either:

- An Outbound message (for Request/Acknowledge operations)
- A Message Function (for Request/Reply operations)

You must first define an adapter datastore in the Designer. Then, the batch job or RTDF can pass a message to one of the adapter operation instances defined in that datastore. To define an adapter, you must:

- Define a datastore object for each adapter instance
- Define one function or one outbound message for each operation instance to which you want to pass a message.

For each adapter instance, define a corresponding datastore object in the Datastore Editor window of the Designer object library.

#### 6.2.2.4.1 To define a JMS adapter datastore

1. From the Datastore Editor:
   a. Select the **Job Server** configured to manage your JMS adapter.
   b. Select the **Adapter instance name** you configured in the Administrator.
2. Select the Adapter Properties tab and enter values for each property.
3. Click **OK** to save values and create the datastore.

### 6.2.2.5 Importing message functions and outbound messages to the datastore

You can pass messages from a batch job or RTDF to an operation instance. Import either a function or an outbound message (depends on the type of operation involved) in the Designer Datastore library for each operation instance.

Real-time data flows use following methods.

| Method | Description |
|---|---|
| Message functions | Pass messages to an operation instance if the RTDF waits for a return XML message from the IR. |
| Outbound messages | Outbound messages Pass messages to an operation instance if the RTDF waits for a confirmation only (not a return XML message) from the IR. |

Operation types in the SAP BusinessObjects Data Services Adapter for JMS have the following invocation types.

| Operation type | Invocation type |
|---|---|
| Request/Reply Operation | Message Function |
| Request/Acknowledge Operation | Outbound Message |

#### 6.2.2.5.1 To import message functions and outbound messages

1. In Designer, double-click the datastore associated with your JMS Adapter Instance to display the Adapter metadata browser window.
2. Right-click the operation instance to be imported and select Import.

   The selected operation instance is added to the datastore.

These message functions and outbound message functions can be used for creating Batch Jobs or RTDFs in SAP BusinessObjects Data Services.

## 6.3 Using the JMS adapter

## 6.3.1 To start an instance of the JMS adapter

1. From the Administrator go to **Adapter Instance** > **Job Server** and select the **Status** tab.
2. Select the check-box next to the previously configured adapter instance.
3. Click **Start**.

When the adapter instance and its operations start, the message “Started” appears in the Status column.

### 6.3.1.1 Operations from SAP BusinessObjects Data Services to the JMS adapter

#### 6.3.1.1.1 Request/Reply - PutGet operation

SAP BusinessObjects Data Services initiates the request by sending a message on a pre-configured request queue. Simultaneously, the software also listens on a pre-configured reply queue. An external JMS-compatible application listening on this request queue, after processing, sends back the response on response queue. This response, in the form of the reply XML message, is returned back to the software.

**Related Topics**

- Testing PutGet: Request/Reply

#### 6.3.1.1.2 Request/Acknowledge - Put operation

SAP BusinessObjects Data Services initiates the request by sending the message on a pre-configured target queue.

**Related Topics**

- Testing Put: Request/Acknowledge

#### 6.3.1.1.3 Request/Acknowledge - PutTopic operation

SAP BusinessObjects Data Services initiates the request by publishing the message to a pre-configured target topic.

**Related Topics**

- Testing PutTopic: Request/Acknowledge

### 6.3.1.2 Operations from Information Resource (IR) to Data Services

#### 6.3.1.2.1 Request/Reply - Get operation

IR initiates the request by putting a message in the source queue of the Get operation. The Get operation receives the message from the source queue during a polling cycle and sends the message to the configured Job service. The service sends a reply message to the Get operation, which then puts the message in the response queue. The IR then gets the message from the response queue.

**Related Topics**

- Testing Get: Request/Reply

#### 6.3.1.2.2 Request/Acknowledge - Get operation

IR initiates the request by putting a message in the source queue of the Get operation. The Get operation receives the message from the source queue during a polling cycle and sends the message to the configured Job service.

**Related Topics**

- Testing Get: Request/Acknowledge

#### 6.3.1.2.3 Request/Acknowledge - GetTopic operation

IR initiates the request by publishing a message to the source topic of the GetTopic operation. The GetTopic operation receives the message from the source queue during a polling cycle and sends the message to the configured Job service.

**Related Topics**

- Testing GetTopic: Request/Acknowledge

## 6.3.2 To run the sample

This section details the JMS adapter operations.

1. Import the `JMSAdapter.atl` file into the Designer. Find the `.atl` file in `<DS_COMMON_DIR>/adapters/jms/samples`. The imported project name is Acta_JMSAdapter_Sample.

2. Change the input and output XML files path for all the batch jobs depending on your location of your <*DS_COMMON_DIR*> environment variable.
3. Use the Administrator Real-Time Services Configuration tab to create the service Queue.TestService referencing job TestService_Job and Topic.TestService referencing job TestServiceTopic_Job.
4. Open Web Administrator and configure a JMS adapter. Define the operations detailed in the following tests.
5. Use the Designer to edit the JMSAdapter datastore and rename it to the name of the adapter you just created.

Before running the sample, create the following queues and topic using your JMS provider utilities:

- Queue.MyQueue
- Queue.ActaQueuePutGet
- Queue.ActaQueuePutGet1
- Queue.ActaQueueGet
- Queue.ActaReplyQueueGet
- Queue.ActaUndeliveredQueue
- Topic.MyTopic

**Note:**
The `JMSAdapterTest.properties` file and the scripts to execute the samples are located in the `<DS_COMMON_DIR>/adapters/jms/samples` directory.

The `JMSAdapterTest.properties` file `TopicConnectionFactoryName` property value is Tcf and the `QueueConnectionFactoryName` property value is Qcf. You must edit this file and change the property values if the adapter was configured using different factory names.

The `JMSAdapterTest.properties` file `MessageSource` property refers to the file `<DS_COMMON_DIR>/adapters/jms/samples/xml/JMSSource.xml`. You must edit this file and change the property value if this is not where your `JMSSource.xml` file is located.

You must edit `setTestEnv.bat` on Windows or `setTestEnv.sh` on UNIX to set the JMS Provider jar files in the class path used by the sample test programs.

### 6.3.2.1 Configuring the JMS provider

Create a JMS Server, Connection Factory and configure JMS queues to run SAP BusinessObjects Data Services Adapter for JMS. For testing the adapter, using sample applications, configure the following queues and topic:

- Queue.MyQueue
- Queue.ActaQueuePutGet
- Queue.ActaQueuePutGet1
- Queue.ActaQueueGet
- Queue.ActaReplyQueueGet
- Queue.ActaUndeliveredQueue

- Topic.MyTopic

Refer to the “Appendix” section for instructions on using Weblogic as the JMS Provider. Steps for JMS Provider may differ from the example provided in this section.

#### 6.3.2.2 To use MQ instead of JNDI configuration

The properties file used by the samples, `JMSAdapterTest.properties`, is set up to use the JNDI configuration. You can edit this file to use MQ configuration parameters.

1. Open the `JMSAdapterTest.properties` file.
2. Set ConfigType = MQ.
3. Set any of the following properties as required by your system:
   - MqQueueManager
   - MqChannel
   - MqComputerName
   - MqPort
   - MqUserID
   - MqPassword
4. For the queue and topic names, use MQ names instead of the JNDI names for the following properties:
   - TopicGetName
   - TopicPutName
   - QueueSourceGetName
   - QueueResponseGetName
   - QueuePutName
   - QueueRequestPutGetName
   - QueueReplyPutGetName

### 6.3.3 Testing PutGet: Request/Reply

To configure the operation type PutGet (Request/Reply), enter the following information in the operation instance configuration page in Administrator.

| Option | Value |
|---|---|
| Operation instance | JMSPutGetOperation |
| Thread count | 1 |
| Display name | JMSPutGetOperation |
| Description | This operation instance represents the PutGet Request/Reply operation. It sends the request message to the request queue and receives the reply message from the reply queue. |
| Enable | true |
| Request queue | Queue.ActaQueuePutGet |
| Reply queue | Queue.ActaQueuePutGet1 |
| Timeout | 200000 |
| Continue after error | true |
| Request format | <*DS_COMMON_DIR*>/adapters/JMS/samples/dtd/JMSPUTGET_SOURCE.dtd |
| Request XML root element | source |
| Reply format | <*DS_COMMON_DIR*>/adapters/JMS/samples/dtd/JMSPUTGET_RESPONSE1.dtd |
| Reply XML root element | source |

After entering this information, click **Apply** and restart the JMS Adapter instance. When the JMS Adapter starts running, the operation instance also starts running.

### Testing on Windows

Open a command prompt window and change directory to <*DS_COMMON_DIR*>\adapters\jms\samples. Run the sample application (external IR) by running sampleTest_PutGet.bat. The application displays the message:

```
Ready to receive message from queue Queue.ActaQueuePutGet
```

Execute the batch Job JMSPutGetOperation_BatchJob from the Designer. This sends the message to the request queue.

The sample application (external IR) listens for a message to arrive at the request queue of the JMSPutGetOperation instance. When it receives the message, it prints a message to the command prompt window such as:

```
Message received:  <?xml version="1.0" encoding="UTF-8"?>  <!-- BusinessObjects Data Services generated XML
 -->  <!-- 2005-05-05.16:41:57(539,223)[1] -->  <source>  <age>18</age>  <salary>200000000</salary>  <ac
no>2356376438743</acno>  </source>
```

The sample test program then sends a reply message to the reply queue configured for the JMSPutGetOperation instance. It echoes a message to the command prompt window such as:

```
Message sent:  <?xml version="1.0" encoding="UTF-8"?>  <source>  <age>ReplyFromJMSIR1</age>  <salary>Re
plyFromJMSIR2</salary>  <acno>ReplyFromJMSIR3</acno>  </source>
```

After the adapter operation receives the reply from the reply queue, it sends the message to the job which then generates the output file `JMSSourceOutput_PutGet.xml` under the directory `<DS_COMMON_DIR>/adapters/JMS/samples/xml`. The contents of the file should be similar to the message sent from the sample test with the addition of a timestamp and error information.

#### Testing on UNIX

Run the sample application (external IR) by running `sampleTest_PutGet.sh` file from the command prompt.

Execute the batch Job JMSPutGetOperation_BatchJob from Designer. This sends the message at the request queue.

Sample application (external IR) listens for the message at the request queue of JMSPutGetOperation instance and sends the message to the reply queue configured for the JMSPutGetOperation instance. After receiving the reply from the reply queue an output file `JMSSourceOutput_PutGet.xml` is generated under the directory `<DS_COMMON_DIR>/adapters/JMS/samples/xml`.

### 6.3.4 Testing PutTopic: Request/Acknowledge

To configure the operation type Put topic (Request/Acknowledge), enter the following information in the operation instance configuration page in the Web Administrator.

| Option | Value |
|---|---|
| Operation instance | JMSPutTopicOperation |
| Thread count | 1 |
| Operation retry count | 5 |
| Operation retry interval | 15000 |
| Display name | JMSPutTopicOperation Display Name |
| Description | JMSPutTopicOperation Display Name |
| Enable | true |
| Destination queue | Topic |

| Option | Value |
|---|---|
| Message format | C:\ProgramFiles\Business Objects\BusinessObjects Data Services |
| Request XML root element | source |
| Persistent message | true |

After entering this information, click **Apply** and restart JMS Adapter instance. When the JMS Adapter starts running, the operation instance also starts running.

Add the testing sections:

#### Testing on Windows

Open a command prompt window and change directory to <*DS_COMMON_DIR*>\adapters\jms\samples. Run the sample application (external IR) by running sampleTest_PutTopic.bat. The application should display the message:

```
Ready to receive message from topic Topic.MyTopic
```

If you do not see this message, then start the JMS publish/subscribe broker. The message should appear after you start the broker.

Execute the batch Job JMSPutTopicOperation_BatchJob from the Designer.

The sample application (external IR) listens for a message to be published by the JMSPutTopicOperation instance. When it receives the message, it will print a message to the command prompt window such as:

```
Received message:  <?xml version="1.0" encoding="UTF-8"?>  <source>  <age>18</age>  <salary>200000000</salary>
  <acno>2356376438743</acno>  </source>
```

After the adapter operation acknowledges sending the message to the IR, the job then generates the output file JMSSourceOutput_PutTopic.xml under the directory <*DS_COMMON_DIR*>/adapters/JMS/samples/xml. The contents of the file should be similar to the message received by the sample test with the addition of a timestamp. Note that this file is created as a result of the design of the job, not as a result of the adapter operation sending a reply message to the job.

#### Testing on UNIX

Run the sample application by running sampleTest_Put.sh file from the command prompt. This sample application listens at the destination queue configured for the Put operation instance.

Execute the batch Job JMSPutOperation_BatchJob from the Designer.

### 6.3.5 Testing Get: Request/Reply

To configure the operation type Get (Request/Reply), enter the following information in the operation instance configuration page in the Web Administrator.

| Option | Value |
|---|---|
| Operation instance | JMSGetOperation |
| Polling interval | 1000 |
| Thread count | 1 |
| Enable | true |
| Source queue | Queue.ActaQueueGet |
| Service | Queue.TestService |
| Timeout | 2000 |
| Continue after error | true |
| Default response queue | Queue.ActaReplyQueueGet |
| Undelivered queue (optional) | Queue.ActaUndeliveredQueue |

After entering this information, click **Apply** and restart the JMS Adapter instance. When the JMS Adapter starts running, the operation instance also starts running.

### Testing on Windows

Run the sample application (external IR) by running `sampleTest_Send.bat` file from the command prompt. This sample application sends the message at the source queue of the Get operation instance configured in the software.

Also, run another sample application (external IR) by running the batch file `sampleTest_Get.bat` file, which receives the reply from SAP BusinessObjects Data Services on a default response queue.

The sample application `sampleTest_Send.bat` (external IR) sends the message as a request on a source queue configured for JMSGetOperation instance. JMSGetOperation instance invokes the real-time batch job and also sends the reply back at the default response queue. The sample application `sampleTest_Get.bat` (external IR) receives the reply on this default response queue. If any error occurs while invoking another service from this Job service, then the original message is sent to the undelivered queue, for reference by the IR.

### Testing on UNIX

Run the sample application (external IR) by running `sampleTest_Send.sh` file from the command prompt. This sample application sends the message at the request queue of the operation instance configured in the software.

Also, run another sample application (external IR) by running the batch file `sampleTest_Get.sh` file. This receives the reply from the software on a default response queue.

The sample application `sampleTest_Send.sh` (external IR) sends the message as a request on a source queue configured for JMSGetOperation instance. JMSGetOperation instance will invoke the real-time batch job and also sends the reply back at the default response queue. The sample application `sampleTest_Get.sh` (external IR) receives the reply on this default response queue. If any error occurs while invoking another service from this Job service, then the error message is sent to the undelivered queue, for reference by the IR.

### 6.3.6 Testing GetTopic: Request/Acknowledge

To configure the operation type Get topic (Request/Acknowledge), enter the following information in the operation instance configuration page in the Web Administrator.

| Option | Value |
|---|---|
| Operation instance | JMSGetTopicOperation |
| Polling interval | 1000 |
| Thread count | 1 |
| Enable | true |
| Source topic | Topic.MyTopic |
| Service | Topic.TestService |
| Timeout | 2000 |
| Continue after error | true |

After entering this information, click **Apply** and restart the JMS Adapter instance. When the JMS Adapter starts running, the operation instance also starts running.

#### Testing on Windows

Run the sample application (external IR) by running the `sampleTest_GetTopic.bat` file from the command prompt. This sample application publishes a message to the source topic of the GetTopic operation instance.

JMSGetTopicOperation, which has subscribed to the topic, receives the message and sends it to the real-time service. The service then puts the message into file `JMSFileTarget_GetTopic.xml` in directory `<DS_COMMON_DIR>/adapters/jms/samples/xml`.

#### Testing on UNIX

Run the sample application (external IR) by running the `sampleTest_GetTopic.sh` file from the command prompt. This sample application publishes a message to the source topic of the GetTopic operation instance.

JMSGetTopicOperation, which has subscribed to the topic, receives the message and sends it to the real-time service. The service then puts the message into file `JMSFileTarget_GetTopic.xml` in directory `<DS_COMMON_DIR>/adapters/jms/samples/xml`.

### 6.3.7 Testing Get: Request/Acknowledge

To configure the operation type Get (Request/Acknowledgment), enter the following information in the operation instance configuration page in the Web Administrator.

| Option | Value |
|---|---|
| Operation instance | JMSGetOperation |
| Polling interval | 1000 |
| Thread count | 1 |
| Enable | true |
| Source queue | Queue.ActaQueueGet |
| Service | Queue.TestService |
| Timeout | 2000 |
| Continue after error | true |
| Default response queue | **Note:**<br>When you specify a value, this operation changes from Request/Acknowledgement to Request/Reply. |
| Undelivered queue | **Note:**<br>When you specify a value, this operation changes from Request/Acknowledgement to Request/Reply. |

After entering this information, click **Apply** and restart the JMS Adapter instance. When the JMS Adapter starts running, the operation instance also starts running.

#### Testing on Windows

Run the sample application by running `sampleTest_Send.bat` file from the command prompt.

This sample application (external IR) sends the message as a request on a source queue configured for JMSGetOperation instance. JMSGetOperation instance invokes the real-time batch job. This creates an output file `JMSSourceOutput_Get.xml` as an acknowledgement at the location `<DS_COMMON_DIR>/adapters/JMS/samples/xml`. No response is sent to the default response queue since it is not configured for this type of operation.

#### Testing on UNIX

Run the sample application by running `sampleTest_Send.sh` file from the command prompt.

This sample application (external IR) sends the message as a request on a source queue configured for JMSGetOperation instance. JMSGetOperation instance invokes the real-time batch job. This creates an output file `JMSSourceOutput_Get.xml` as an acknowledgement at the location `<DS_COMMON_DIR>/adapters/JMS/samples/xml`. No response is sent to the default response queue since it is not configured for this type of operation.

## 6.3.8 Testing Put: Request/Acknowledge

To configure the operation type Put (Request/Acknowledge), enter the following information in the operation instance configuration page in the Web Administrator.

| Option | Value |
|---|---|
| Operation instance | JMSPutOperation |
| Thread count | 1 |
| Display name | JMSPutOperation |
| Description | This operation instance represents the Put Request/Acknowledge operation. It queues the message to the configured destination queue. |
| Enable | true |
| Destination queue | Queue.MyQueue |
| Request format | `<DS_COMMON_DIR>/adapters/JMS/samples/dtd/JMSPUT_SOURCE.dtd` |
| Request XML root element | source |

Click **Apply** after entering this information, then restart the adapter instance.

When the JMS Adapter is running, the operation instance is also running.

### Testing on Windows

Open a command prompt window and change directory to `<DS_COMMON_DIR>\adapters\jms\samples`. Run the sample application (external IR) by running `sampleTest_Put.bat`. The application should display the message:

```
Ready to receive message from queue Queue.MyQueue.
```

Execute the batch Job JMSPutOperation_BatchJob from the Designer.

The sample application (external IR) listens for a message to arrive at the request queue of the JMSPutOperation instance. When it receives the message, it will print a message to the command prompt window such as:

```
Received message:  <?xml version="1.0" encoding="UTF-8"?>  <source>  <age>18</age>  <salary>200000000</salary>
  <acno>2356376438743</acno>  </source>
```

After the adapter operation acknowledges sending the message to the IR, the job then generates the output file `JMSSourceOutput_Put.xml` under the directory `<DS_COMMON_DIR>/adapters/JMS/samples/xml`. The contents of the file should be similar to the message received by the sample test with the addition of a timestamp. Note that this file is created as a result of the design of the job, not as a result of the adapter operation sending a reply message to the job.

#### Testing on UNIX

Run the sample application by running `sampleTest_Put.sh` file from the command prompt. This sample application listens at the destination queue configured for the Put operation instance.

Execute the batch Job JMSPutOperation_BatchJob from the Designer.

The sample application receives the message from the destination queue and an output file `JMSSourceOutput_Put.xml` as an acknowledgment gets created under the directory `<DS_COMMON_DIR>/adapters/JMS/samples/xml`.

## 6.3.9 Technical implementation

### 6.3.9.1 Design considerations

In the current design:

- JMS queues and topics used in the Operation instances must be pre-configured in the Messaging System.
- Only XML messages are handled.
- GetTopic operations should be configured to specify a Thread Count of 1. Since each thread would be a subscriber to the topic, each thread would receive the same message and send it to the service, resulting in multiple copies of the same message going to the service.

### 6.3.9.2 Error handling and tracing

Error messages are logged in error log file under the `<DS_COMMON_DIR>/adapters/log` directory before throwing any exception. The name of the error log file is same as the name of the adapter configured in the Administrator.

For tracing, the trace messages are logged in the trace file under the `<DS_COMMON_DIR>/adapters/log` directory. The name of the trace file is same as the name of the adapter configured in the Administrator. You can enable the trace option in the Administrator for this adapter.

Trace message shows the execution flow of the adapter and contain useful information on finding the cause of an error. The output in this trace file is of great help for SAP Business User Support.

## 6.4 Appendix

### 6.4.1 Weblogic as JMS provider

Before you run the SAP BusinessObjects Data Services Adapter for JMS, you need to create a JMSServer, Connection Factory and configure JMS queues.

- Create a JMS Server
- Start the BEA Weblogic server.
- Open the Weblogic console.
- Under services\JMS, click **Servers**.
- Click Create a new JMS Server button.

Create the instance of JMS server. Then, click **Create**.

Click the Target link on the screen and select the server from available block to a chosen block. Click **Apply** to create the server instance.

#### 6.4.1.1 To create a JMS Connection Factory

1. Start the BEA Weblogic server
2. Open the Weblogic console
3. Under services\JMS, click **Connection Factories**.

Configure the Connection Factory. For testing purposes, “JMSConnections.AdapterConnectionFactory” must be configured.

Click the Target link on the screen. Select the server from available block to chosen block.

#### 6.4.1.2 To configure the JMS Connection Factory

For testing purposes, “JMSConnections.AdapterConnectionFactory” must be configured.

1. Click the Target link on the screen.
2. Select the server from available block to chosen block.
3. Click **Apply** to create the connection factory.

#### 6.4.1.3 To create a JMS queue

1. Start the BEA Weblogic server.
2. Open the Weblogic console
3. Under services\JMS\Servers\ConfigJMSServer\Destinations, click **Create a New JMS Queue**.

For testing purposes, configure the following queues in the server:

- Queue.MyQueue
- Queue.ActaQueuePutGet
- Queue.ActaQueuePutGet1
- Queue.ActaQueueGet
- Queue.ActaReplyQueueGet
- Queue.ActaUndeliveredQueue

# Using the HTTP adapter

## 7.1 Introduction

### 7.1.1 Audience and assumptions

This section assumes the following:

- You understand how to use the Designer to design and run Data Services data flows.
- You have a basic understanding of how to use the Administrator to administer Data Services processes. (To use an adapter, you administer it from the Administrator.)
- You have a working knowledge of the environment this adapter is targeting.
- You know the role an adapter plays in business systems integration
- You have some familiarity with the XML markup language and XML configuration schemas.
- Because you will integrate Data Services and an external system, some familiarity with systems administration and systems integration issues is recommended.

### 7.1.2 About this section

This section tells you how to use the HTTP Adapter for integrating SAP BusinessObjects Data Services with external applications using the HTTP protocol.

This section provides a detailed description of installing the HTTP Adapter. This includes the descriptions of the pre-requisite software along with their supported versions, the details of the adapter components, the environment setup for both Data Services and Tomcat and instructions for executing the adapter.

## 7.2 Overview

### What does the HTTP Adapter provide?

The HTTP Adapter provides data transfer using HTTP and HTTPS protocols.

### What is HTTP protocol?

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990 and its use has increased steadily over the years, mainly because it has proven useful as a generic middleware protocol.

The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server.

The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta information, and possible entity-body content.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80 [19], but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used.

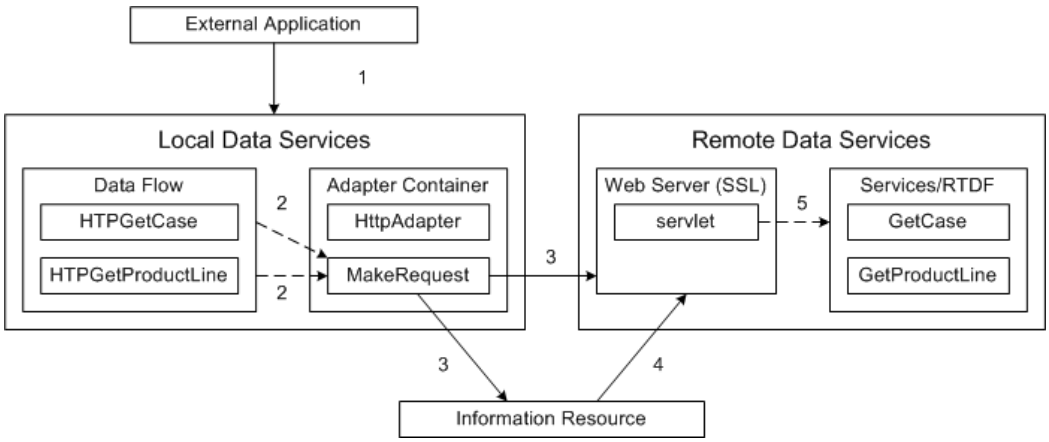
### Scope of the HTTP Adapter

The major scope of the HTTP Adapter is:

- To ensure rapid integration of diverse systems and applications by making use of HTTP protocol with the SAP BusinessObjects Data Services platform to meet unique business process requirements saving valuable time and effort.
- The HTTP Adapter supports SSL (Secure Socket Layer) to implement security over the HTTP protocol. Using HTTPS protocol, the data will be protected from any unscrupulous elements.
- The HTTP Adapter supports compress type data encoding while sending and receiving the information and thus saves network traffic.
- Request/Reply and Request/Acknowledge services by SAP BusinessObjects Data Services can be initiated through this adapter.

## 7.3 Architecture

The following diagram shows the functional overview of the HTTP Adapter. It depicts two remote SAP BusinessObjects Data Services installations using the HTTP Adapter to exchange information. The same diagram applies for interaction between SAP BusinessObjects Data Services and any other third-party software supporting the HTTP protocol.




The flow of control is explained below. Refer to the above diagram.

1. External application invokes a service on SAP BusinessObjects Data Services.
2. A data flow makes a call to the adapter operation instance.
3. The operation instance receives the XML data from RTDF and makes a request on the remote Data Services server. The operation instance forms the request URL by reading its configuration file. The URL contains servlet name and the service name, which are configured as part of the operation instance configuration. In the case of requesting to the information resource, a resource-specific URL will be configured as part of the operation instance configuration.
4. The information resource (for example, Siebel) can make a request on the remote Data Services server by using HTTP or HTTPS protocol. The information resource forms the URL, which contains the servlet name and service name.
5. The servlet runs on the HTTP server (Tomcat) that is a part of Data Services. This HTTP server can be SSL enabled, depending on user requirements. The servlet processes the request to get the service name and XML data. It will then invoke that service running locally in Data Services and send the reply back to the client.

## 7.4 Installation and configuration

The HTTP Adapter is installed with every SAP BusinessObjects Data Services Job Server.

You need to create and configure an adapter instance and necessary adapter operations of the HTTP Adapter. Adapter operations identify the integration options available for the configured adapter instance. The HTTP Adapter servlet is also installed while installing the HTTP Adapter.

The operations provided with the HTTP Adapter include the following:

- Request/Reply operation

  This operation is used to execute a remote Data Services service in the Request/Reply mode. It makes the request to the remote Data Services machine where the HTTP Adapter servlet is running

and waits for the reply. The service name to be executed can be configured as part of the URL while configuring the operation instance.

- Request/Acknowledge operation

  This operation is used to execute a remote Data Services service in the Request/Acknowledge mode. It makes the request to the remote Data Services machine where the HTTP Adapter servlet is running and does not wait for the reply. Instead, it sends acknowledgement if the operation is successful. The service name to be executed can be configured as part of the URL while configuring the operation instance.

### 7.4.1 To configure the HTTP Adapter

To use the HTTP Adapter, create an instance of the adapter together with a real-time data flow (RTDF) or data flow that you define using the Designer. Before using an HTTP Adapter instance, you must first configure it as explained in this section. To configure the HTTP Adapter means to configure one or more instances of the adapter and one or more operation instances for each adapter instance.

All SAP BusinessObjects Data Services adapters communicate with Data Services through a designated Adapter Manager Job Server. An adapter must first be installed on the same computer as this Job Server before you can integrate the adapter with Data Services using the Administrator and Designer. After the adapter is installed:

1. Use the Server Manager utility to configure adapter connections with the Adapter Manager Job Server. For details, see the “To configure Job Servers” section of the *Administrator's Guide*.
2. Open the Administrator and:
   a. Add at least one instance of the adapter to the Data Services system.
   b. If the adapter instance includes operations, add at least one operation for each adapter instance.
   c. Start the adapter instance.

      Operations are started automatically.
3. Open the Designer and create an adapter datastore. Use metadata accessed through the adapter to create batch and/or real-time jobs. For details, see the “Adapter datastores” section of the *Designer Guide*.

### 7.4.2 To configure an HTTP Adapter instance

Use the Administrator to add an HTTP Adapter to the SAP BusinessObjects Data Services system and to edit existing adapter configurations. Until you add the adapter in the Administrator, you cannot run jobs using information from that adapter.

To add an adapter instance in the administrator:

1. Select **Adapter Instances** > **Job Server**.
2. Click the **Configuration** tab.
3. Click **Add**.
4. Select the HTTP Adapter from the list of those available on the Job Server.
5. Enter the required information to create an HTTP Adapter instance.
   For details, see “Adapter instance configuration information”.
6. Click **Apply**.

The Administrator adds the adapter instance to the list of those available to the Data Services system.

### 7.4.2.1 Adapter instance configuration information

Complete the following fields in the Administrator to set up an HTTP Adapter instance in the SAP BusinessObjects Data Services system:

| Field | Configuration information |
|---|---|
| Adapter instance name | Enter a unique name that identifies this instance of the HTTP Adapter. |
| Access Server host | Enter the host ID of the computer running the Access Server that connects to this adapter instance. |
| Access Server connection port | The Access Server host's message broker port. After you log into the Administrator for this Access Server, select **Configuration** > **Interfaces** to view message broker port information. |
| Classpath | All adapter Java programs require specific JAR files in the CLASSPATH to use when starting `javaw.exe`. For example:<br>• `<LINK_DIR>/lib/acta_adapter_sdk.jar`<br>• `<LINK_DIR>/lib/acta_broker_client.jar`<br>• `<LINK_DIR>/lib/acta_tool.jar`<br>• `<LINK_DIR>/ext/lib/xerces.jar`<br>• `<LINK_DIR>/lib/acta_http_adapter.jar`<br>• `<LINK_DIR>/lib/jcert.jar`<br>• `<LINK_DIR>/lib/jnet.jar`<br>• `<LINK_DIR>/lib/jsse.jar` |
| Autostart | When set to True, the adapter interface automatically starts when the Administrator starts. |

| Field | Configuration information |
|---|---|
| Trace mode | Set this flag to control the number of trace messages the adapter writes. There are two settings:<br>• True: The adapter interface writes additional information messages to help debug problems.<br>• False: The adapter interface writes minimal information messages. The adapter writes trace messages to the *adapter_instance_name*_trace.txt file in the <*DS_COMMON_DIR*>/adapters/logs directory. |
| Application command line parameters | Additional command line parameters used for the javaw.exe command line and for the adapter itself. See specific adapter documentation for details. |
| Adapter type name | The name of the adapter used to create this instance. (Read-only) |
| Adapter version | The version of the adapter used to create this instance. (Read-only) |
| Adapter class | A name that identifies the adapter class. The name depends on the type of adapter. (Read-only) |
| Keystore password | Required if requests are made using the HTTPS protocol. If a password is given, it is used to check the integrity of the keystore data. Otherwise, the integrity of the keystore is not checked. |

### 7.4.3 To configure an operation instance

Use the Administrator to add an operation instance to an adapter instance.

1. Select **Adapter Instance > Job Server**
2. Click the **Configuration** tab.
3. Click **Operations** under Dependent Objects.
4. Click **Add** to configure a new operation or click the link of an existing operation to edit its configuration.
5. Select an operation type from the list and click **Apply**. The options that appear on this page depend on the operation-specific design.
6. Complete the operation instance configuration form. For details, see “Operation instance configuration information”.
7. Click **Apply**.

#### 7.4.3.1 Operation instance configuration information

### Request/Reply operation configuration

Complete the following fields in the Administrator to set up a Request/Reply operation instance:

| Field | Configuration information |
|---|---|
| Operation instance | Enter the unique operation instance name. In the Designer, Request/Reply operation metadata object will be imported with this name. |
| Thread count | The number of copies of the Request/Reply operation to run in parallel. For parallel (asynchronous) processing of messages coming from a real-time service, use more than one copy. If the sequency of messages is important (synchronous processing), you should not use more than one thread. The default is 1.<br>**Note:**<br>Multiple copies of real-time services must be supported by multiple copies of Request/Reply. |
| Display name | Enter the operation instance display name. This display name will be visible in the Designer's metadata browsing window. |
| Description | Enter the operation instance description. This description will be visible in the Designer's metadata browsing window. |
| Enable | True if the Adapter SDK will start this operation instance when the adapter starts, otherwise false. |
| Target URL | URL where you want to send the HTTP request. Data Services uses the following server URL format:<br>`http://host:port/admin/servlet/com.acta.adapter.http.server.HTTPActaServlet?ServiceName=ServiceName`<br>• Host: The IP address or host name of the Access Server.<br>• Port: The port number of the Access Server.<br>• ServiceName: The name of the service. |
| Request method | The HTTP request method to be used for submitting the request. The possible values are POST and GET. |
| Content-Type | This is used to set the content type header of the request. It specifies the nature of the data by giving type and subtype identifiers. |
| Content-Language | The ISO code for the language in which the request's document is written. For example, `en` means that the language is English in one of its forms. |
| Content-Encoding | Specifies the encoding mechanism used for sending the request. Currently only x-compress and x-gzip are used. |
| Continue if untrusted | Specifies whether to continue the operation if the HTTP server is untrusted when using the HTTPS protocol.<br>• True: The operation instance will continue for untrusted servers.<br>• False: The operation instance will be terminated for untrusted servers. |

| Field | Configuration information |
|---|---|
| Request DTD | The DTD file name that defines the request XML message used in this operation. |
| Request XML root element | The name of the XML root element in the request DTD. |
| Reply DTD | The DTD file name that defines the reply XML message used in this operation. |
| Reply XML root element | The name of the XML root element in the reply DTD. |

**Note:**
Restart the HTTP Adapter instance so that all configuration changes take effect.

### Request/Acknowledge operation configuration

Complete the following fields in the Administrator to configure a Request/Acknowledge operation instance:

| Field | Configuration information |
|---|---|
| Operation instance | Enter the unique operation instance name. In the Designer, Request/Acknowledge operation metadata object will be imported with this name. |
| Thread count | The number of copies of the Request/Acknowledge operation to run in parallel. For parallel (asynchronous) processing of messages coming from a real-time service, use more than one copy. If the sequency of messages is important (synchronous processing), you should not use more than one thread. The default is 1.<br>**Note:**<br>Multiple copies of real-time services must be supported by multiple copies of Request/Acknowledge. |
| Display name | Enter the operation instance display name. This display name will be visible in the Designer's metadata browsing window. |
| Description | Enter the operation instance description. This description will be visible in the Designer's metadata browsing window. |
| Enable | True if the Adapter SDK will start this operation instance when the adapter starts, otherwise false. |
| Target URL | URL where you want to send the HTTP request. Data Services uses the following server URL format:<br>`http://host:port/admin/servlet/com.acta.adapter.http.server.HTTPActaServlet?ServiceName=ServiceName`<br>• Host: The IP address or host name of the Access Server.<br>• Port: The port number of the Access Server.<br>• ServiceName: The name of the service. |

| Field | Configuration information |
|---|---|
| Request method | The HTTP request method to be used for submitting the request. The possible values are POST and GET. |
| Content-Type | This is used to set the content type header of the request. It specifies the nature of the data by giving type and subtype identifiers. |
| Content-Language | The ISO code for the language in which the request document is written. For example, `en` means that the language is English in one of its forms. |
| Content-Encoding | Specifies the encoding mechanism used for sending the request. Currently only x-compress and x-gzip are used. |
| Continue if untrusted | Specifies whether to continue the operation if the HTTP server is untrusted when using the HTTPS protocol.<br>• True: The operation instance will continue for untrusted servers.<br>• False: The operation instance will be terminated for untrusted servers. |
| Request DTD | The DTD file name that defines the request XML message used in this operation. |
| Request XML root element | The name of the XML root element in the request DTD. |

**Note:**
Restart the HTTP Adapter instance so that all configuration changes take effect.

### 7.4.4 Defining the adapter datastore

Use the HTTP Adapter with an RTDF/data flow when the the RTDF/data flow passes a message to an operation instance, using either:

- An outbound message (for Request/Acknowledge operations)
- A message function (for Request/Reply operations)

You must define an adapter datastore in the Designer. An RTDF/data flow can then pass a message to one of the adapter's operation instances defined in the datastore.

To define an adapter datastore, you must:

- Define a datastore object for each adapter instance.
- Define one function or one outbound message for each operation instance to which you want to pass a message.

The following sections summarize the Designer tasks involved. For complete information, see the *Designer Guide*.

### 7.4.4.1 To define an adapter datastore

For each adapter instance, define a corresponding datastore object in the Designer object library, in the "Datastore Editor" window.

1. In the **Job Server** box, select the Job server that is configured to handle your HTTP adapter.
2. For **Adapter instance name**, choose the instance name you configured in the Administrator.
3. Click to select the Adapter Properties tab and enter values for each property.
4. Click **OK** to save values and create the datastore.

### 7.4.4.2 To import message functions and outbound messages

Messages can be passed from a real-time data flow (RTDF) to an operation instance. You must import either a function or an outbound message (depending on the type of operation involved) in the Designer Datastore library for each operation instance.

Real-time dataflows use:

| Method | Description |
|---|---|
| Message functions | Pass messages to an operation instance if the RTDF waits for a return XML message from the information resource. |
| Outbound messages | Outbound messages pass messages to an operation instance if the RTDF waits for a confirmation only (not a return XML message) from the information resource. |

The operation types in the HTTP Adapter have the following invocation types:

| Operation type | Invocation type |
|---|---|
| Request/Reply | Message function |
| Request/Acknowledge | Outbound message |

To import the message functions and outbound messages:

1. In the Designer, double click on the datastore that has an associated HTTP Adapter instance.
2. In the "Adapter Metadata Browser" window, right-click on the operation instance to import and select **Import**.

The selected operation instance will be added to the datastore. These message functions and outbound messages can be used for creating the RTDF/data flow in Data Services.

## 7.4.5 Configuring SSL with the HTTP adapter

With Secure Sockets Layer (SSL), the HTTP Adapter can use secure transport over the TCP/IP network.

### Server-side configuration

To use SSL with the HTTP adapter, you must properly configure your web application server for SSL support.

If you are using the default web application server bundled with SAP BusinessObjects BI platform, see the *SAP BusinessObjects BI Platform Administrator's Guide* for information on how to configure SSL.

If you are using a different third-party web application server, see the web application server's documentation.

### Client-side configuration

On the client side, the HTTP Adapter client internally handles the details of certificate authentication by implementin the X509TrustManager interface and using SSLSocketFactory classes from the HttpsURLConnection class.

Whenever an HTTPS request is made to the SSL-enabled web server, the client requests the server's certificate, which may be issued by a standard authority, such as VeriSign. If the HTTP client finds the certificate to be one that is trusted by comparing it to the certificate store in `<LINK_DIR>/ext/jre/lib/security`, it retrieves all data from the web server. In the case of an un-trusted certificate, the HTTP client throws an SSLException to the caller.

The HTTP client requires the password for querying the local keystore for verification. This password can be supplied through the `keystorePassword` parameter specified as a part of the adapter configuration.

The operation instance will read the configurable **Continue if untrusted** flag and, based on its value, trust the unknown server and its certificate. If the parameter is set to False, then the SSLException is shown to the user with a friendly message and logged in Data Services' trace files and the client does not retrieve any data from the server. If the parameter is set to True, then the SSLException is logged in Data Services' error and trace files and the client proceeds to retrieve data from the server. The certificate file `untrust.cer` is downloaded to the user's current working directory or to the `<LINK_DIR>/bin` directory. This certificate file can later be imported into the JDK certificate keystore by using the `keytool` command-line utility:

```
keytool -import -alias <description> -file untrust.cer -keystore
<full_path_of_cacerts_file> cacerts -storepass changeit
```

## 7.5 Using the HTTP Adapter

To start the adapter instance:

1. Select the **Status** tab under "Adapter Instances" in the Administrator.
2. Check the **Select** box next to the adapter instance that you configured previously.
3. Click **Start**.

When your adapter instance and its operations start, the message “Started” appears in the status column. To confirm that all operations are started, click **Operations** in the "Dependent Objects" column.

If you have a real-time service set up on your system, you can invoke it through the HTTP interface:

http://localhost:8080/admin/jsp/InvokeService.jsp

Using this interface, you can invoke the selected service by sending the input XML to the HTTP Adapter servlet running on the remote machine where the service is configured.

**Note:**
For information about how to set up a test service, see the “Verifying real-time connectivity” section in the *Installation Guide*.

#### Request/Reply from Data Services

The Request/Reply operation instance sends the request to the remote SAP BusinessObjects Data Services machine and waits for the reply.

To check the sample Request/Reply operation, see the “Testing the Request/Reply operation” section.

#### Request/Acknowledge from Data Services

The Request/Acknowledge operation instance sends the message to the remote SAP BusinessObjects Data Services machine and gives an acknowledgement.

To check the sample Request/Acknowledge operation, see the “Testing the Request/Acknowledge operation” section.

### 7.5.1 Testing the Request/Reply operation

To configure the Request/Reply operation for testing, use the following information to configure the operation instance in the Administrator:

| Field | Configuration information |
|---|---|
| Operation instance | HTTP_ReqReply_Function |
| Thread count | 1 |
| Display name | HTTP_ReqReply_Function |
| Description | Performs the Request/Reply operation |
| Enable | true |

| Field | Configuration information |
|---|---|
| Target URL | For HTTP operation, use:<br>http://*ds_host_name*:*access_server_port*/admin/servlet/com.acta.adapter.http.server.HTTPActaServlet?ServiceName=Test<br>For HTTPS operation, use:<br>https://*ds_host_name*:*tomcat_https_port*/admin/servlet/com.acta.adapter.http.server.HTTPActaServlet?ServiceName=Test<br>**Note:**<br>By default, the HTTPS port of the Tomcat server is 8443. This can be changed in the Tomcat configuration file (`acta-server.xml` on Windows, and `acta-server1.xml` on UNIX) |
| Request method | Post |
| Content-Type | text/xml |
| Content-Language | en |
| Content-Encoding | application/nocompress |
| Continue if untrusted | true |
| Request DTD | <*DS_COMMON_DIR*>/adapters/Http/samples/dtd/HTTPTestIn.dtd |
| Request XML root element | test |
| Reply DTD | <*DS_COMMON_DIR*>/adapters/Http/samples/dtd/HTTPTestOut.dtd |
| Reply XML root element | test |

After configuring the operation instance, click **Apply** and then restart the HTTP Adapter instance. After the HTTP Adapter is running, the operation instance will also be in a running state.

Execute the `HTTP_ReqRep_BatchJob` job in the Designer.

After the batch job executes successfully, an output file `OutputRep.xml` will be created in the <*DS_COMMON_DIR*>/adapters/Http/samples/xml directory.

## 7.5.2 Testing the Request/Acknowledge operation

To configure the Request/Acknowledge operation for testing, use the following information to configure the operation instance in the Administrator:

| Field | Configuration information |
|---|---|
| Operation instance | HTTP_ReqAck_Outbound |
| Thread count | 1 |
| Display name | HTTP_ReqAck_Outbound |
| Description | Performs the Request/Acknowledge operation |
| Enable | true |
| Target URL | For HTTP operation, use:<br>http://*ds_host_name*:*access_server_port*/admin/servlet/com.actaadapter.http.server.HTTPActaServlet?ServiceName=Test<br>For HTTPS operation, use:<br>https://*ds_host_name*:*tomcat_https_port*/admin/servlet/com.actaadapter.http.server.HTTPActaServlet?ServiceName=Test<br>**Note:**<br>By default, the HTTPS port of the Tomcat server is 8443. This can be changed in the Tomcat configuration file (`acta-server.xml` on Windows, and `acta-server1.xml` on UNIX) |
| Request method | Post |
| Content-Type | text/xml |
| Content-Language | en |
| Content-Encoding | application/nocompress |
| Continue if untrusted | true |
| Request DTD | <*DS_COMMON_DIR*>/adapters/Http/samples/dtd/HTTPTestIn.dtd |
| Request XML root element | test |

After configuring the operation instance, click **Apply** and then restart the HTTP Adapter instance. After the HTTP Adapter is running, the operation instance will also be in a running state.

Execute the `HTTP_ReqAck_BatchJob` job in the Designer.

After the batch job executes successfully, an output file `OutputAck.xml` will be created in the <*DS_COMMON_DIR*>/adapters/Http/samples/xml directory.

## 7.6 Error handling and tracing

All error and trace messages are logged to the log files in the <*DS_COMMON_DIR*>/adapters/log directory. The names of the error and trace log files match the names of the adapter instance as configured in the Administrator, and appended with _error.txt for error logs and _trace.txt for trace logs.

For example, if the name of the HTTP Adapter instance is “HTTPAdapter”, the name of the error file will be HTTPAdapter_error.txt and the name of the trace file will be HTTPAdapter_trace.txt.

# Using the SuccessFactors adapter

## 8.1 About this section

This section tells you how to create and configure a SuccessFactors adapter.

### 8.1.1 Overview

You can create a SuccessFactors adapter instance in the Management Console of the Administrator.

Once you create an adapter instance and a datastore, you can browse and import SuccessFactors database tables to use as a source or a target in a Data Services dataflow.

### 8.1.2 Audience and assumptions

This section assumes the following:

- You understand how to use the Designer to design and run Data Services data flows.
- You have a basic understanding of how to use the Administrator to administer Data Services processes. (To use an adapter, you administer it from the Administrator.)
- You have a working knowledge of the environment this adapter is targeting.
- You know the role an adapter plays in business systems integration
- You have some familiarity with the XML markup language and XML configuration schemas.
- Because you will integrate Data Services and an external system, some familiarity with systems administration and systems integration issues is recommended.

## 8.2 Installation and configuration

The SuccessFactors adapter is installed with every Data Services job server. In order to use the adapter, you need to create and configure an adapter instance.

**Tip:**
In the SAP BusinessObjects Data Services Service Manager, make sure the **Support adapter, message broker and SNMP communication** option is enabled. This option can be found in the "Job Server Properties" window.

### 8.2.1 Configure the SuccessFactors adapter instance

Use the Administrator to add an adapter for SuccessFactors to the Data Services system and to edit adapter configurations. Until you add the adapter in the Administrator, you cannot run jobs using information from that adapter.

**Note:**
Before you add an adapter in the Administrator, you must establish Administrator connection to your adapter-enabled repository. For more information about connecting repositories to the Administrator, see the “Administrator Management” section of the *Management Console Guide*.

1. Select a job server under the **Adapter Instances** node in the navigation tree.
2. Click on the **Adapter Configuration** tab.
3. Click **Add** to see a list of adapters managed by the job server.
4. Select the adapter for SuccessFactors.
5. Complete the information on the **Adapter instance startup configuration** page.

| Option | Description |
|---|---|
| **Adapter Instance Name** | (Required) Enter a unique name to identify this instance of the adapter. Spaces are not allowed. |
| **Access Server Host** | You can leave this blank. |
| **Access Server Port** | You can leave this blank.. |
| **Use SSL Protocol** | When set to True, the adapter instance uses SSL (Secure Sockets Layer) protocol when routing data over the Internet. |
| **Character Set** | Converts text characters to and from bytes for data. |
| **Metadata Character Set** | Converts text characters to and from bytes for metadata. |
| **Adapter Retry Count** | Applies if the adapter instance fails or crashes. Enter 0 for no retries and a negative number for indefinite retries. |
| **Adapter Retry Interval** | The wait, in milliseconds, between adapter retry attempts. |
| **Classpath** | Indicates the -classpath Java parameter value when the adapter starts. |
| **Autostart** | When set to True, the adapter interface automatically starts when the Administrator starts. |

| Option | Description |
|---|---|
| **Trace mode** | Set this flag to control the number of trace messages the adapter writes. When set to True, the adapter interface writes additional information messages to help debug problems.<br>When set to False, the dapter interface writes minimal information messages. The adapter writes trace message to the *adapter_instance_name_trace*.txt file in the %DS_COMMON_DIR%\adapters\logs directory. |
| **Additional Java Launcher Options** | Enabled when launching the Java process that hosts the adapter.<br>If you are connecting to the adapter from behind a proxy server, add the following to the end of the Additional Java Launcher options:<br>-Dhttps.proxyHost=*proxy_server_name* -Dhttps.proxyPort=*proxy_server_port* |
| **Adapter type name** | (Read-only) The name of the adapter used to create this instance. |
| **Adapter version** | (Read-only) The version of the adapter used to create this instance. |
| **Adapter Class** | (Read-only) A name that identifies the adapter class. The name depends on the type of adapter. |

6. Click **Apply**.

## 8.2.2 Start and stop the adapter instance

Click the **Status** tab to view the status of all adapter instances you configured. From this tab, you can start adapter instances and shut down or abort instances that are running.

**Note:**
If you make any configuration changes to the SuccessFactors adapter, you'll need to restart the adapter instance before the changes will take effect.

From the **Status** tab, you can also navigate to view Adapter Instance configuration details, Log Files, and Dependent Objects for each configured adapter instance.

## 8.2.3 Create a SuccessFactor adaptor datastore

To associate the SuccessFactors adaptor with a data flow, you must create an adapter datastore in the Designer. For more information about creating an adapter datastore, see the “Datastores” section of the *Designer Guide*.

1. In the **Datastores** tab of the Designer Object Library, right-click and select **New**.
2. In the "Datastore Editor" window, type a unique name in the **Datastore name** box.
3. Select **Adaptor** from the **Datastore type** list.
4. Select the job server associated with the SuccessFactors adaptor.
5. Choose the name of the adaptor instance from the **Adapter instance name** list.
6. Click **Advanced** and configure the following options:
   - Web service end point
   - Company ID
   - Username
   - Password
   - Default Base64 binary field length in kilobyte (KB)

     Binary data is encode in ASCII using Base64 format and Data Services stores this ASCII data in a varchar field. You must specify the size for the Data Services varchar field. The default is 16 KB.
7. Click **OK** to save values and finish creating the datastore.

   If you do not provide the correct user name and password, or if you entered an invalid parameter, you will see an error message stating that the adapter connection failed.

### 8.2.4 Browse and import metadata

You can view and import SuccessFactors tables to use as a source or a target in your dataflows. For general information on how to browse and import metadata using a Data Services datastore, see the “Datastores” section of the *Designer Guide*.

To view data:

1. Double-click the adapter datastore icon.

   The "Adapter Metadata Browser" window opens with a list of table objects (and their descriptions) from SuccessFactors that are available for viewing.
2. Right-click on a table name and select **Import** or right-click on a datastore and select **Import By Name**. For more information about importing metadata, see the *Designer Guide*.

   Once imported, the table appears under the SuccessFactors datastore.
3. To open an imported table, double-click the table icon. You can then view input and output schemas, and so on.

#### 8.2.4.1 Metadata mapping

SuccessFactors data types map to Data Services data types as follows:

| SuccessFactors data types | Data Services data types | Description |
|---|---|---|
| Integer | int | Integer value. |
| Long | decimal(20,0) | |
| Float | double | Double values. |
| Double | double | Double values. |
| String | varchar | Character strings. SuccessFactors provides the size. Data is in UTF-8. |
| Boolean | varchar(5) | Boolean true/false value. |
| Date | date | Date values in *YYYY-MM-DD* format |
| Datetime | datetime | The date/time values that the adapter retrieves from SuccessFactors are all in ISO 8601 format (*YYYY-MM-DDThh:mm:ssZ*). When the adapter communicates datetime information to Data Services, it receives those values in local time and the time zone field is not considered. |
| Binary | varchar | In Base64 format. Size is defined in a datastore parameter named Default Base64 binary field length. |

### 8.2.5 Use SuccessFactor tables as a source or a target in your dataflow

You can use a SuccessFactors table as a source or a target in a dataflow. For more information about how to do this, see the “Source and target objects” section in the *Designer Guide*.

### Source information

The following adapter source options are available:

| Option | Description |
|---|---|
| Batch size | Specifies the number of rows to be processed as a batch. |
| Column delimiter | Separates data between columns. |
| Row delimiter | Separates data between rows. |

Data Services uses the SuccessFactors Query Language specification to push down operations to the source or target database. The following rules apply:

- Only columns can be in the projection.
- The Where clause is of the form *<column> operation <constant>*.
- Only columns are allowed in the ORDER BY clause.

### Target information

Each SuccessFactors table has an id field. The id field is an internal key. When a row is inserted, SuccessFactors creates an id for that row. When inserting and upserting rows, make sure the input data does not include the id field. If the data does include the id field, SuccessFactors returns an error.

When a row is updated or deleted, SuccessFactors requires the id field to be present in the input data. The id field is used to identify a row. If the id field is not present, SuccessFactors returns an error.

The following adapter target option are available:

| Option | Description |
|---|---|
| Batch size | Specifies the number of rows to be processed as a batch. |
| Column delimiter | Separates data between columns. |
| Row delimiter | Separates data between rows. |

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Use auto correct</td><td>Checks the target table for existing rows before adding new rows to the table. Note that using this option can slow jobs.<br><br>When you set this parameter to true, Data Services does the following:<br><br>
<table>
<tr><th>Row status</th><th>Action</th></tr>
<tr><td>Insert</td><td>Inserts a row if it doesn't already exist. If the row does exist, the software updates the row.</td></tr>
<tr><td>Update</td><td>When there is no id field in the input data, the existing row is updated. If the row doesn't exist, the row is inserted. When the table has an id field, the row is updated.</td></tr>
<tr><td>Delete</td><td>Deletes the row.</td></tr>
</table>
</td></tr>
<tr><td>Use audit</td><td>Logs data for auditing. Data Services creates audit files and stores them in the <code>%DS_COMMON_DIR%\log\SFSF</code> directory. The format of the file is <Datastore name>_<Table name>_<Process id><Thread id><code>.dat</code>.<br><br>If you set the Use audit option to true, Data Services logs data for the following scenarios:<br>• If there is no user input keys and a rows cannot be deleted or update, the id field is automatically logged.<br>• User input keys are always logged. For insert row, if you do not specify an input keys, an error is returned. You can specify input keys in the Query transform that is connected to the SuccessFactors loader.</td></tr>
</table>

## 8.2.6 Manually add the SuccessFactors certificate

Data Services automatically includes the SuccessFactors certificate in its Java keystore so that Data Services recognizes the SuccessFactors instance as a trusted website.

However, if there is an error regarding the SuccessFactors certificate, you can manually add the certificate back into the Java keystore.

1. Download the certificate file from the SuccessFactors web site.
2. For Windows operating systems, do the following at the command prompt:
   a. Type `set JAVA_HOME=%LINK_DIR%\ext` and press Enter.
   b. Type `set path=%LINK_DIR%\ext\jre\bin;%path%` and press Enter.

c. Type `cd %LINK_DIR%\ext\jre\lib\security` and press Enter.
d. Type `keytool -import -alias sfsf -file "your certificate file" -keystore cacerts` and press Enter.

**Note:**
The default keystore password is changeit. When the prompt asks if you can trust the certificate, type Yes.

3. For UNIX operating systems, do the following at the shell prompt:
   a. Type `export JAVA_HOME=$LINK_DIR/ext` and press Enter.
   b. Type `export PATH=$LINK_DIR/ext/jre/bin:$PATH` and press Enter.
   c. Type `cd $LINK_DIR/ext/jre/lib/security` and press Enter.
   d. Type `keytool -import -alias <certificate file> -keystore cacerts` and press Enter.

**Note:**
The default keystore password is changeit. When the prompts asks if you can trust the certificate, type Yes.

# Object creation XML toolkit

## 9.1 Overview

The object creation XML toolkit is a collection of utilities and features that enable you to programmatically create objects such as jobs, dataflows, and workflows from your own application and then import, validate, and execute them in SAP BusinessObjects Data Services.

The toolkit consists of several primary components:

| Component | Purpose |
|---|---|
| XML schema | Used to defines Data Services objects |
| Designer import and export tools | Used to create template or example XML of objects |
| Web services | Used to send externally generated objects to Data Services and import, validate, or execute them |
| Auxiliary utilities | Used to perform specific tasks required by some jobs, such as datastore password encryption |

## 9.2 Using the toolkit

In general, the object creation XML toolkit is suitable for two scenarios:

- Exporting Designer-created object definitions for use as templates that are customized with substitution variables
- Exporting Designer-created object definitions for use as a guideline for creating new object definitions from scratch

For both scenarios, we recommend that you use the Designer and its import and export capabilities heavily until you have the object definition to generate from within your own application. After you have suitable object defintions, you can use web services to import, validate, and execute jobs within Data Services.

The recommended workflow for using the toolkit is:

1. Create templates of your objects in the Designer.
2. Export your objects from the Designer to the XML format.
3. Customize the exported XML objects for generation in your external application.
4. Import, validate, and execute the XML objects generated by your application through Data Services' web services.

### 9.2.1 Templating objects

A key feature of the object creation XML toolkit is the ability to use the Designer to template your objects. You can use the Designer to create and debug objects, and then export them to an XML format for customization and use in your own external application. To create a template object for use in the object creation XML toolkit, the process is the same as creating any other normal object in the Designer.

**Tip:**
Although it is possible to use only the XML schema to write complete repository objects from scratch, we strongly recommend that you use the Designer to create your objects due to complex interactions between many parts of the XML structure.

For more information about using the Designer to create and configure jobs, dataflows, transforms, and other objects, see the *Designer Guide*.

For more information about configuration options available for specific objects, see the *Reference Guide*.

### 9.2.2 Exporting objects

The Designer allows you to export SAP BusinessObjects Data Services objects in a standardized, reproducible XML format. When exporting objects, you can choose to export an entire job or individual objects. Each export option produces a single XML file that contains all exported objects. If you want to create separate XML files for each object, use the export editor to individually export each object.

For more information about using the export editor, see the “Export/Import” section of the *Administrator Guide*.

### 9.2.3 Adapting objects

Because the options of many XML objects, such as transforms, have complex interactions and do not always match the way the configuration appears in the Designer, it's strongly recommended that you adapt exported versions of these objects that have already been mostly configured how you want. By

adapting existing objects, you can be more confident that the objects generated by your application will be free of problems, and function as you expect.

For example, you can configure several transforms using the Designer and then export them to XML files. In your application, you can assemble the pre-configured transforms into working dataflows, and use web services to import and run them with Data Services.

Exported objects can still be customized through the use of parameters, as well as simply modifying the XML directly.

**Related Topics**

- Parameters and variables

## 9.2.4 Using web services

After generating custom objects in your application, you can use web service operations provided with SAP BusinessObjects Data Services to import, validate, and execute them.

The general process for using the object creation XML toolkit with web service operations has several steps:

1. Log in

   If web service security is enabled, use the Logon operation and get a session object.

2. Import objects

   Use the Import_Repo_Object operation to import objects to the repository. You can import single or multiple objects at a time. However, for large XML, you may need to import objects one at a time.

   If an error occurs during importing, the operation returns the error message.

3. Validate objects

   After the objects have been imported successfully, use the Validate_Repo_Object to perform a semantic validation. You can also perform the validation on only the highest level object to recursively validate all dependent objects.

   If an error occurs during validation, the operation returns the error message.

4. Execute objects

   After the objects have been validated successfully, use the Run_Batch_Job operation to execute the job now stored in the repository. Use the job name as the parameter for the execution request.

   If an error occurs during execution, use the returned runID code and the Get_Error_Log and Get_Trace_Log operations to retrieve messages specific to this execution of the job.

5. Retrieve operational metadata

   Use the runID code returned by Run_Batch_Job along with the Get_Error_Log, Get_Trace_Log, and Get_Monitor_Log operations to retrieve errors, warnings, trace messages, and performance statistics for this execution of the job.

6. Remove objects (optional)

If you don't want the objects to persist in the repository after execution, use the Delete_Repo_Objects operation to remove them. Because the operation does not remove dependencies automatically, you need to call it once for each object you want to remove.

7. Remove operational metadata

   Error and trace logs for non-execution requests are automatically cleared at the completion of each request. However, execution logs are not automatically cleared, and must be cleaned up manually. You can schedule the cleanup of execution logs by setting an appropriate log retention period in the Administrator.

   For more information, see the *Management Console Guide*.

8. Log out

   If you used the Login operation to take advantage of web services security, use the Logout operation to log out and end the session.

**Related Topics**

- Logon
- Import_Repo_Object
- Validate_Repo_Object
- Run_Batch_Job
- Get_Error_Log
- Get_Trace_Log
- Get_Monitor_Log
- Delete_Repo_Objects
- Logout

## 9.2.5 Encrypting passwords

For security, passwords for things such as datastore connections are encrypted when stored in the repository. However, when importing objects, SAP BusinessObjects Data Services does not perform the encryption operation. Because of this, in order to use passwords in externally generated objects, you must use the encrypted form of the password in the generated XML.

To encrypt passwords outside of Data Services, use the `al_encrypt` utility included with the object creation XML toolkit. You can use `al_encrypt` with parameters for either an existing keyfile or a plain-text passphrase.

By default, `al_encrypt` outputs the encrypted password to the screen. However, you can use output redirection to store the encrypted password to a file, and then read the password from that file when generating the XML for your object. For example:

```
al_encrypt -e mypassword -p thepassphrase > c:\password.txt
```

For complete syntax information for `al_encrypt`, see the *Administrator's Guide*: Command line administration, Password encryption.

## 9.2.6 Best practices

When using the object creation XML toolkit, there are a number of best practices that you can follow to simplify your workflow and minimize any problems that you may encounter.

### Importing objects

- While you are allowed to import multiple objects at once, you may need to import objects individually when the XML syntax is large.
- Objects should be imported into the repository in order. That is, lower level objects in the dependency chain should be imported before higher level objects. For example, if you have a job that contains a dataflow that uses a file format, you should import the file format, followed by the dataflow, and then the job. By properly maintaining the correct import order, you can avoid cross-referencing issues.
- You can avoid import problems by validating your generated XML before importing. That is, ensuring that elements are closed correctly, and so on.

### Validating objects

- Validate your objects using the Designer during your design phase. The validation web service performs only a runtime validation that is not as comprehensive as the Designer's validation. Validating in the Designer can provide you with more detailed information that may be helpful in resolving issues.
- You can validate objects individually or recursively by validating a high level object.

### Other

- All object definitions must be placed within the correct container element, DataIntegratorExport. You can place one or more object definitions into the DataIntegratorExport element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<DataIntegratorExport repositoryVersion="12.2.0.0000" productVersion="12.1.0.0">

   <!-- One or more object definitions -->

</DataIntegratorExport>
```

- Some Data Services objects support expressions. For example, you can assign a query to the output field of a query transform. In addition, scripts and custom functions are defined as expressions. The expressions used in these objects are represented twice in exported XML:
  1. As an expr attribute with the same format as shown in the Designer
  2. As a complex XML hierarchy

  When importing these objects, only the format contained in the expr attribute is required.
- For Data Quality transform custom configurations, it is not required to have best practice input and output fields defined.

## 9.2.7 Limitations

### General limitations

You should be aware of several general limitations when using the object creation XML toolkit:

- Some parts of object validation can only be performed in the Designer user interface.

  To more quickly identify and fix validation problems, you can import and validate your objects in the Designer when you are developing custom objects.

- The web service operations do not support import or bulk loading of Data Cleanse dictionaries.

  Before using other web services with the Data Cleanse transform, you must configure the appropriate dictionaries with the Designer.

### Concurrent use issues

When multiple users access the same repository concurrently, the Data Services engine locks appropriate tables and serializes requests so that the repository is always in a structurally valid state. However, you should be aware of other issues that can arise due to concurrent repository access:

- Referential integrity violation due to object removal

  The Delete_Repo_Objects web service operation does not enforce the rule that an object cannot be deleted if it is being used by other objects. The exception is a datastore, which cannot be deleted if it contains any child objects.

  You should maintain referential object integrity in your application outside Data Services.

- Interference between web service operations

  Some available web service operations have the potential to interfere with others if they are currently running. For example, a job could fail if the job is deleted with the Delete_Repo_Objects operation at the same time it is being executed with the Run_Batch_Job operation.

  You should be careful to synchronize web service operations so that conflicts do not occur.

One way to avoid most concurrent use issues is by maintaining good naming conventions. For example, by using unique names for objects created in different instances of your application, you can effectively create a segregated domain for each application instance. As a result, the application instances will not interfere with each other when performing operations such as creating, modifying, or reading repository objects.

## 9.3 XML schema reference

The object creation XML toolkit supports the creation and usage of all objects available in Data Services. This section provides information about the structure and usage of some of the most common objects. For objects not covered in detail, use the export function of the Designer and the “Objects” section in the *Reference Guide* as a guide for creating and using your own objects.

**Caution:**
All examples provided in this section are for reference only. Do not attempt to run them.

#### Enclosing objects to import

To import XML content into the Data Services repository, it must be enclosed in a DataIntegratorExport element. Specify the repository and product versions using the repositoryVersion and productVersion attributes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<DataIntegratorExport repositoryVersion="12.2.0.0000"
  productVersion="12.2.0.0">

</DataIntegratorExport>
```

The DataIntegratorExport element is required only once per XML file, and should contain all objects that you want to import.

## 9.3.1 Batch job

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

#### Creating the object definition

Open the batch job definition with the DIJob element.

```
<DIJob name="JobName" typeId="2">
```

Specify steps within the job using the DISteps and DICallStep elements.

```
<DISteps>
  <DICallStep typeID="1" calledObjectType="Dataflow"
    name="DataflowName"></DICallStep>
</DISteps>
```

Apply job-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="job_checkpoint_enabled" value="no" />
  <DIAttribute name="job_collect_statistics" value="no" />
  <DIAttribute name="job_collect_statistics_monitor"
    value="no" />
  <DIAttribute name="job_enable_assemblers" value="yes" />
  ...
</DIAttributes>
```

End the job definition with the closing tag of the DIJob element.

```
</DIJob>
```

### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording used in the Designer. For a complete description of each attribute, see the "Descriptions of objects" section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| job_checkpoint_enabled | |
| job_collect_statistics | **Job properties > Execution options > Collect statistics for optimization** |
| job_collect_statistics_monitor | **Job properties > Execution options > Collect statistics for monitoring** |
| job_enable_assemblers | |
| job_enable_audit | |
| job_enable_dataquality | |
| job_export_repo | |
| job_export_reports | |
| job_isrecoverable | |
| job_mode | |
| job_monitor_sample_rate | **Job properties > Execution options > Monitor sample rate** |
| job_name | **Job properties > General > Name** |
| job_print_version | |
| job_testmode_enabled | |
| job_trace_abapquery | **Job properties > Trace > Trace ABAP Query** |
| job_trace_all | **Job properties > Execution options > Print all trace messages** |
| job_trace_ascomm | **Job properties > Trace > Access Server Communication** |
| job_trace_assemblers | **Job properties > Trace > Assemblers** |
| job_trace_audit | **Job properties > Trace > Audit Data** |
| job_trace_dataflow | **Job properties > Trace > Data Flow** |
| job_trace_idoc_file | **Job properties > Trace > IDoc file reader** |
| job_trace_memory_loader | **Job properties > Trace > Memory Target** |

| DIAttribute name | Designer |
|---|---|
| job_trace_memory_reader | **Job properties > Trace > Memory Source** |
| job_trace_optimized_dataflow | **Job properties > Trace > Optimized Dataflow** |
| job_trace_parallel_execution | **Job properties > Trace > Trace Parallel Execution** |
| job_trace_rfc_function | **Job properties > Trace > RFC Function** |
| job_trace_row | **Job properties > Trace > Row** |
| job_trace_script | **Job properties > Trace > Scripts and Script Functions** |
| job_trace_session | **Job properties > Trace > Session** |
| job_trace_sql_only | **Job properties > Trace > SQL Only** |
| job_trace_sqlfunctions | **Job properties > Trace > SQL Functions** |
| job_trace_sqlloaders | **Job properties > Trace > SQL Loaders** |
| job_trace_sqlreaders | **Job properties > Trace > SQL Readers** |
| job_trace_sqltransforms | **Job properties > Trace > SQL Transforms** |
| job_trace_stored_procedure | **Job properties > Trace > Stored Procedure** |
| job_trace_table | **Job properties > Trace > Tables** |
| job_trace_table_reader | **Job properties > Trace** |
| job_trace_transform | **Job properties > Trace > Transform** |
| job_trace_userfunction | **Job properties > Trace** |
| job_trace_usertransform | **Job properties > Trace** |
| job_trace_workflow | **Job properties > Trace > Work Flow** |
| job_type | |
| job_use_statistics | **Job properties > Execution options > Use collected statistics** |

| DIAttribute name | Designer |
|---|---|
| locale_codepage | |
| locale_language | |
| locale_territory | |

## 9.3.2 Workflow

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

### Creating the object definition

Open the workflow definition with the DIWorkflow element.

```
<DIWorkflow name="WorkflowName" typeId="2">
```

Specify steps within the workflow using the DISteps, DICallStep, and DIScript elements.

```
<DISteps>
  <DIScript>
    <DIUIOptions>
      <DIAttribute name="ui_display_name"
        value="ScriptName" />
    </DUIOptions>
    ...
  </DIScript>
  <DICallStep typeID="1" calledObjectType="Dataflow"
    name="DataflowName"></DICallStep>
  <DIScript>
    <DIUIOptions>
      <DIAttribute name="ui_display_name"
        value="ScriptName" />
    </DUIOptions>
    ...
  </DIScript>
</DISteps>
```

Apply workflow-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="run_once" value="no" />
  <DIAttribute name="unit_of_recovery" value="no" />
</DIAttributes>
```

End the workflow definition with the closing tag of the DIWorkflow element.

```
</DIWorkflow>
```

#### Available DIAttribute names

The available DIAttribute names correspond to workflow parameters and properties, but do not necessarily match the wording used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| run_once | **Properties > General > Execute only once** |
| unit_of_recovery | **Properties > General > Recover as a unit** |

### 9.3.3 Dataflow

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

#### Creating the object definition

Open the dataflow definition with the DIDataflow element.

```
<DIDataflow name="DataflowName" typeId="1">
```

Invoke transforms within DIDataflow by using the DITransforms element.

```
<DITransforms>
  <DIFileSource typeId="33" formatName="FileFormatName"
    filename="filename.txt"/>
  <DIQuery typeId="22"/>
  <DIFileTarget typeId="3"/>
  ...
</DITransforms>
```

Apply dataflow-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="Cache_type" value="pageable_cache" />
  <DIAttribute name="Parallelism_degree" value="0" />
  <DIAttribute name="allows_both_input_and_output"
    value="yes" />
  <DIAttribute name="run_once" value="no" />
  <DIAttribute name="use_dataflow_links" value="no" />
  <DIAttribute name="use_datastore_links" value="yes" />
  <DIAttribute name="validation_xform_exists" value="no" />
  <DIAttribute name="validation_xform_stats" value="no" />
</DIAttributes>
```

End the dataflow definition with the closing tag of the DIDataflow element.

```
</DIDataflow>
```

#### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording used in the Designer. For a complete description of each attribute, see the "Descriptions of objects" section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| Cache_type | **Properties > General > Cache type** |
| Parallelism_degree | **Properties > General > Degree of parallelism** |
| allows_both_input_and_output | |
| run_once | **Properties > General > Execute only once** |
| use_dataflow_links | |
| use_datastore_links | **Properties > General > Use database links** |
| validation_xform_exists | |
| validation_xform_stats | |

## 9.3.4 Script

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

#### Creating the object definition

Open the script definition with the DIScript element.

```
<DIScript>
```

Specify the name of the script with a DIAttribute element in the DIOptions block.

```
<DIOptions>
  <DIAttribute name="ui_display_name" value="script_name" />
</DIOptions>
```

Include expressions with the DIExpression element. In exported objects, additional XML syntax may be present. However, this additional syntax is optional; only the expr attribute of DIExpression is required.

```
<DIExpression isString="true" expr="script_expression">
  <Additional optional XML syntax/>
</DIExpression>
```

Include functions with the DIFunctionCallStep element. Define the function using a DIExpression element. In exported objects, additional XML syntax may be present. However, this additional syntax is optional.

```
<DIFunctionCallStep typeId="23">
  <DIExpression isString="true" expr="function_call">
    <Additional optional XML syntax/>
  </DIExpression>
</DIFunctionCallStep>
```

End the script definition with the closing tag of the DIScript element.

```
</DIScript>
```

#### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| ui_display_name | **Properties > General > Name** |

For more information about scripts, see the “Objects, Script” section in the *Reference Guide*.

## 9.3.5 File format

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

#### Creating the object definition

Open the file format definition with the DIFlatFileDatastore element.

```
<DIFlatFileDatastore name="FileFormatName" typeId="3">
```

Specify field names, types, and sizes with DIElement elements in the DISchema block.

```
<DISchema>
  <DIElement name="FieldName1" datatype="VARCHAR" size="7">
  </DIElement>
  <DIElement name="FieldName2" datatype="VARCHAR" size="7">
  </DIElement>
  ...
</DISchema>
```

Apply file format-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="abap_file_format" value="no" />
  <DIAttribute name="blank_pad" value="leading" />
  <DIAttribute name="cache" value="yes" />
```

```
  <DIAttribute name="column_delimiter" value=", " />
  ...
</DIAttributes>
```

End the file format definition with the closing tag of the DIFlatFileDatastore element.

```
</DIFlatFileDatastore>
```

### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording used in the Designer. For a complete description of each attribute, see the "Descriptions of objects" section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| abap_file_format | **File Format Editor > General > Type > SAP transport** |
| adaptable | **File Format Editor > General > Adaptable Schema** |
| beginning_of_file_string | **File Format Editor > Input/Output > BOF Marker** |
| blank_pad | **File Format Editor > Default format > Blank padding** |
| cache | |
| column_delimiter | **File Format Editor > Delimiters > Column** |
| column_width | |
| column_width*n* | **File Format Editor > Field Size** for column *n* |
| data_alignment | **File Format Editor > General > Data Alignment** |
| date_format | **File Format Editor > Default Format > Date** |
| datetime_format | **File Format Editor > Default Format > Date-Time** |
| end_of_file_string | **File Format Editor > Input/Output > EOF Marker** |
| escape_character | **File Format Editor > Default Format > Escape char** |
| file_format | |
| file_location | **File Format Editor > Data File(s) > Location** |
| file_name | **File Format Editor > Data File(s) > File name(s)** |
| file_type | **File Format Editor > General > Type** |
| ignore_row_markers | **File Format Editor > Default Format > Ignore row marker(s)** |
| locale_codepage | **File Format Editor > Locale > Code page** |
| locale_language | **File Format Editor > Locale > Language** |
| locale_territory | |
| name | **File Format Editor > General > Name** |
| null_indicator | **File Format Editor > Default Format > NULL inicator** |

| DIAttribute name | Designer |
|---|---|
| number_of_rows_to_skip | **File Format Editor > Input/Output > Skipped rows** |
| number_of_threads | **File Format Editor > General > Parallel process threads** |
| reader_capture_data_conversion_errors | **File Format Editor > Error handling > Capture data conversion errors** |
| reader_capture_row_format_errors | **File Format Editor > Error handling > Capture row format errors** |
| reader_error_file_name | **File Format Editor > Error handling > Error file name** |
| reader_error_file_root_dir | **File Format Editor > Error handling > Error file root directory** |
| reader_log_data_conversion_warnings | **File Format Editor > Error handling > Log data conversion warnings** |
| reader_log_row_format_warnings | **File Format Editor > Error handling > Log row format warnings** |
| reader_maximum_warnings_to_log | **File Format Editor > Error handling > Maximum warnings to log** |
| reader_write_error_rows_to_file | **File Format Editor > Error handling > Write error rows to file** |
| root_dir | **File Format Editor > Data File(s) > Root directory** |
| row_delimiter | **File Format Editor > Delimiters > Row** |
| skip_row_header | **File Format Editor > Input/Output > Skip row header** |
| table_weight | |
| time_format | **File Format Editor > Default Format > Time** |
| transfer_argument | **File Format Editor > Custom Transfer > Arguments** |
| transfer_custom | **File Format Editor > General > Custom transfer program** |
| transfer_name | **File Format Editor > Custom Transfer > Program executable** |
| transfer_password | **File Format Editor > Custom Transfer > Password** |
| transfer_user | **File Format Editor > Custom Transfer > User name** |
| use_root_dir | |
| write_bom | **File Format Editor > Input/Output > Write BOM** |
| write_row_header | **File Format Editor > Input/Output > Write row header** |

#### 9.3.5.1 To use as a source

To use a file format as a source within a dataflow, invoke the format with the DIFileSource element. The DIFileSource element must be placed within the DITransforms section of a dataflow.

```
<DIFileSource typeId="33" formatName="FormatName"
  filename="FileName">
```

Define the name of the source with a DIAttribute element.

```
<DIUIOptions>
<DIAttribute name="ui_display_name" value="SourceName" />
</DIUIOptions>
```

Specify a name for the output schema with the DIOutputView element. By default, it is set to the format name. However, you can change it to any unique string, and use it in all downstream transforms.

```
<DIOutputView name="SchemaName" />
```

Apply source-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="adaptable" value="no" />
  <DIAttribute name="cache" value="yes" />
  <DIAttribute name="connection_port" value="no" />
  <DIAttribute name="file_location" value="local" />
  ...
</DIAttributes>
```

End the source definition, with the closing tag of the DIFileSource element.

```
</DIFileSource>
```

### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| adaptable | **Source File Editor > General > Adaptable Schema** |
| cache | **Source File Editor > General > Cache** |
| connection_port | |
| file_location | **Source File Editor > Data File(s) > Location** |
| name | **Source File Editor > General > Name** |
| reader_capture_data_conversion_errors | **Source File Editor > Error handling > Capture data conversion errors** |
| reader_capture_row_format_errors | **Source File Editor > Error handling > Capture row format errors** |
| reader_filename_col | **Source File Editor > Source information > Column name** |
| reader_filename_col_size | **Source File Editor > Source information > Column size** |

| DIAttribute name | Designer |
|---|---|
| reader_filename_only | **Source File Editor > Source information > Include path** |
| reader_include_filename | **Source File Editor > Source information > Include file name column** |
| reader_log_data_conversion_warnings | **Source File Editor > Error handling > Log data conversion warnings** |
| reader_log_row_format_warnings | **Source File Editor > Error handling > Log row format warnings** |
| reader_maximum_warnings_to_log | **Source File Editor > Error handling > Maximum warnings to log** |
| reader_write_error_rows_to_file | **Source File Editor > Error handling > Write error rows to file** |
| root_dir | **Source File Editor > Data File(s) > Root directory** |
| table_weight | |
| transfer_custom | **Source File Editor > General > Custom transfer program** |

### 9.3.5.2 To use as a target

To use a file format as a target within a dataflow, invoke the format with the DIFileTarget element. The DIFileTarget element must be placed within the DITransforms section of a dataflow.

```
<DIFileTarget typeId="3" formatName="FormatName"
  filename="FileName">
```

Define the name of the target with a DIAttribute element.

```
<DIUIOptions>
<DIAttribute name="ui_display_name" value="TargetName" />
</DIUIOptions>
```

Specify a name for the input schema with the DIInputView element. By default, it is set to the format name. However, you can change it to any unique string, and use it in all downstream transforms.

```
<DIInputView name="SchemaName" />
```

Apply target-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="connection_port" value="no" />
  <DIAttribute name="file_location" value="local" />
  <DIAttribute name="isstreamdebugfile" value="no" />
  <DIAttribute name="loader_load_choice" value="replace" />
  ...
</DIAttributes>
```

End the target definition, with the closing tag of the DIFileTarget element.

```
</DIFileTarget>
```

#### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| connection_port | **Target File Editor > General > Make port** |
| file_location | **Target File Editor > Data File(s) > Location** |
| isstreamdebugfile | |
| loadler_load_choice | |
| name | **Target File Editor > General > Name** |
| root_dir | **Target File Editor > Data File(s) > Root directory** |
| transfer_custom | **Target File Editor > General > Custom transfer program** |
| validate_decimal_data | **Target File Editor > Default Format > Validate decimal data** |

## 9.3.6 Database datastore

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

#### Creating the object definition

Open the database datastore definition with the DIDatabaseDatastore element.

```
<DIDatabaseDatastore name="datastore_name" typeId="3">
```

Apply database datastore-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="DBLiveLoad" value="no" />
  <DIAttribute name="application_type" value="Custom" />
  <DIAttribute name="cdc_enabled" value="no" />
  <DIAttribute name="datastore_repotype" value="local" />
  ...
  <DIAttribute name="ds_configurations" hasNestedXMLTree="true">
```

Specify one or more datastore configurations inside the DIAttribute block with a DSConfigurations block. Only one configuration can be set as default.

```
    <DSConfigurations>
      <DSConfiguration default="true" name="configuration_name">
        <case_sensitive>no</case_sensitive>
        <database_type>Microsoft_SQL_Server</database_type>
        <loader_xact_size>1000</loader_xact_size>
        <locale_codepage>default</locale_codepage>
        <locale_language>default</locale_language>
        <locale_territory>default</locale_territory>
        <mssql_windows_authentication>
          no
        </mssql_windows_authentication>
        <password>;907A8897CEF453232929BD93946</password>
        <server_codepage>default</server_codepage>
        <sql_server_database>
          DS32_Source
        </sql_server_database>
        <sql_server_dataserver>
          testMachine
        </sql_server_dataserver>
        <sql_server_version>
          Microsoft SQL Server 2000
        </sql_server_version>
        <user>ods</user>
      </DSConfiguration>
    </DSConfigurations>
```

Close the DIAttribute block.

```
  </DIAttribute>
</DIAttributes>
```

End the database datastore definition with the closing tag of the DIDatabaseDatastore element.

```
</DIDatabaseDatastore>
```

#### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. The specific DIAttributes and XML elements available vary greatly depending on which type of database you are accessing. It's recommended that you create your datastore definitions using the Designer and then export them to XML.

For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

## 9.3.7 Database table

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

#### Creating the object definition

Open the database table definition with the DITable element.

```
<DITable name="TableName" owner="Owner"
  datastore="DatastoreName" database="DatabaseName"
  description="Description">
```

Apply table-specific properties with a DIProperties block. To define the table as a template table, set the value of the Loader_Is_Template_Table DIAttribute to yes.

```
<DIProperties>
  <DIAttribute name="Table_Type" value="TABLE" />
  <DIAttribute name="Estimated_Row_Count" value="50000" />
  <DIAttribute name="Loader_Is_Template_Table" value="no" />
  <DIAttribute name="db_alias_name" value="ODS" />
  ...
</DIProperties>
```

Specify column definitions with DIColumn elments. You can also define the content type for a column with the Content_Type attribute.

```
<DIColumn name="Cust_ID" datatype="VARCHAR" size="10"
  nullable="false" />
<DIColumn name="Cust_classf" datatype="VARCHAR" size="2"
  nullable="true" />
<DIColumn name="Address" datatype="VARCHAR" size="35"
  nullable="true" Content_Type="ADDRESS">
  <DIProperties>
    <DIAttribute name="Content_Type" value="ADDRESS" />
  </DIProperties>
</DIColumn>
```

Define the primary keys with a DIPrimaryKey block.

```
<DIPrimaryKey>
  <DIPrimaryKeyColumn name="Cust_ID" />
</DIPrimaryKey>
```

Define the unique table index with a DITableIndex element and specify the column name in a DIIndexColumn element.

```
<DITableIndex name="PK__ODS_CUSTOMER__7C8480A" unique="true">
  <DIIndexColumn name="Cust_ID" />
</DITableIndex>
```

End the database table definition with the closing tag of the DITable element.

```
</DITable>
```

#### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

### 9.3.7.1 To use as a source

To use a database table as a source within a dataflow, invoke the table with the DIDatabaseTableSource element. The DIDatabaseTableSource element must be placed within the DITransforms section of a dataflow.

```
<DIDatabaseTableSource typeId="22" datastoreName="DatastoreName"
  ownerName="OwnerName" tableName="TableName">
```

Specify a name for the output schema with the DIOutputView element.

```
<DIOutputView name="SchemaName" />
```

Apply source-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="array_fetch_size" value="1000" />
  <DIAttribute name="cache" value="yes" />
  <DIAttribute name="connection_port" value="no" />
  <DIAttribute name="enable_partitioning" value="no" />
  ...
</DIAttributes>
```

End the source definition, with the closing tag of the DIDatabaseTableSource element.

```
</DIDatabaseTableSource>
```

### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| array_fetch_size | **Source Table Editor > Performance > Array fetch size** |
| cache | **Source Table Editor > Performance > Cache** |
| connection_port | **Source Table Editor > Make port** |
| enable_partitioning | |
| name | **Source Table Editor > Datastore name** |
| package_size | |
| reader_is_DB2CDC_table | |
| reader_overflow_file | |

| DIAttribute name | Designer |
|---|---|
| reader_template_table | |
| reader_use_overflow_file | |
| table_weight | **Source Table Editor > Performance > Join rank** |

### 9.3.7.2 To use as a target

To use a database table as a target within a dataflow, invoke the table with the DIDatabaseTableTarget element. The DIDatabaseTableTarget element must be placed within the DITransforms section of a dataflow.

```
<DIDatabaseTableTarget typeId="11" bulkLoader="false"
  datastoreName="DatastoreName" ownerName="OwnerName"
  tableName="TableName">
```

Specify a name for the input schema with the DIInputView element.

```
<DIInputView name="SchemaName" />
```

Apply target-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="connection_port" value="no" />
  <DIAttribute name="loader_template_table" value="yes" />
  ...
  <DIAttribute name="ldr_configuration_enabled"
    value="yes" />
  <DIAttribute name="ldr_configurations"
    hasNestedXMLTree="true">
```

Specify the target configuration inside the DIAttribute block with a LDRConfigurations block.

```
    <LDRConfigurations>
      <LDRConfiguration database_type="Microsoft_SQL_Server"
        database_version="Microsoft SQL Server 2000">
        <auto_correct_using_merge>Yes</auto_correct_using_merge>
        <bulk_ldr_all_rows></bulk_ldr_all_rows>
        <bulk_ldr_max_errors></bulk_ldr_max_errors>
        <bulk_ldr_rows_per_commit></bulk_ldr_rows_per_commit>
        <enable_partitioning>no</enable_partitioning>
        <ignore_column_case>yes</ignore_column_case>
        <ignore_columns_null>No</ignore_columns_null>
        <ignore_columns_value></ignore_columns_value>
        <loader_auto_correct>no</loader_auto_correct>
        <loader_bulk_load>0</loader_bulk_load>
        ...
      </LDRConfiguration>
    </LDRConfigurations>
```

Close the DIAttribute block.

```
  </DIAttribute>
</DIAttributes>
```

End the target definition, with the closing tag of the DIDatabaseTableTarget element.

```
</DIDatabaseTableTarget>
```

#### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. The specific DIAttributes and XML elements available vary greatly depending on which type of database you are accessing. It's recommended that you create your database table targets using the Designer and then export them to XML.

For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| connection_port | **Target Table Editor > Target > Make port** |
| name | **Target Table Editor > Target > Datastore name** |
| use_unicode_varchar | |

## 9.3.8 Data Quality transforms

The options of Data Quality transforms have complex interactions and do not always match the way the transform configuration appears in the Designer. It's strongly recommended that you adapt exported versions of these transforms that have already been mostly configured how you want. By adapting existing objects, you can be more confident that the objects generated by your application will be free of problems, and function as you expect.

Because of the complexity of the Data Quality transforms, XSD files are provided for each transform. By default, the XSD files are installed to `LINK_DIR\Admin`.

**Note:**
XML exported from the Designer may not validate correctly against the provided XSD files due to element ordering. However, any objects created by using the XSD files will operate correctly when imported to SAP BusinessObjects Data Services.

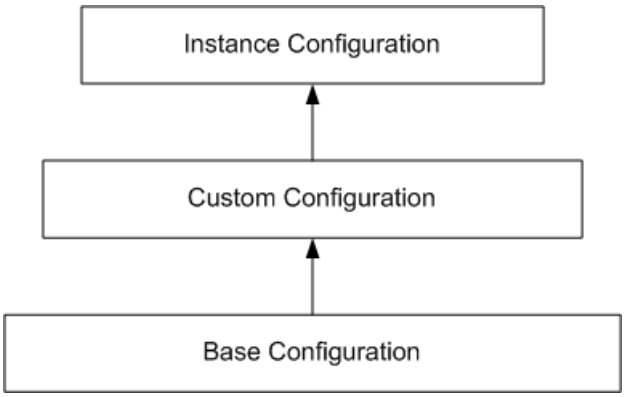
#### Related Topics

- Adapting objects

### 9.3.8.1 Hierarchy and inheritance

The ability to reuse transform configurations is a powerful feature of all Data Quality transforms available in SAP BusinessObjects Data Services. To successfully use Data Quality transforms with the object creation XML toolkit, it's important to understand the rules of option inheritance and configuration reuse.

Data Quality transforms support three levels of configurations: base configuration, custom configuration, and instance configuration. Each level of configuration inherits the options and settings from the levels below it. However, settings defined explicitly at a higher level always take precedence over those inherited from a lower level.




In the Designer, the base configuration is simply represented in the form of the base transform. For example, the Global_Address_Cleanse transform. The base configuration carries the default settings for all options available for that transform. If you double-click any base transform, the configuration is opened in the Transform Configuration Editor. The Options tab lists all options set on the base configuration.

The custom configuration inherits the default settings from the base configuration, but also provides the ability to modify the values of any options at the custom level. In the Designer, the custom configuration level is represented by the available transform configurations. For example, for Global Address Cleanse, custom transform configurations such as Australia_AddressCleanse, Europe_Ad dressCleanse, and USA_AddressCleanse are available. For more information about transform configurations, see the “Data Flows, Transforms” section in the *Designer Guide*.

The instance configuration inherits default settings from the custom configuration, and provides the ability to further modify the values of any options. In the Designer, the instance configuration is represented by a transform instance within a data flow.

## 9.3.9 Query transform

**Tip:**
It's strongly recommended that you use the Designer to create and export your objects, and then use your application to adapt them as required. Many object options and parameters interact in complex ways, and do not always match the way they appear in the Designer.

### Creating the object definition

Open the query transform definition with the DIQuery element.

```
<DIQuery typeId="122">
```

Specify the name of the query transform and additional user interface options, such as the SQL where clause, in the DIUIOptions block.

```
<DIUIOptions>
  <DIAttribute name="ui_display_name" value="QryCDC" />
  <DIAttribute name="ui_where_text" value="WhereClause" />
</DIUIOptions>
```

Define the output schema with DIElement elements within a DISchema block. The input schema for each field is defined in the ui_mapping_text attribute, using the format *InputSchemaName.InputFieldName*.

```
<DISchema name="QryCDC">
  <DIElement name="CUST_ID" datatype="VARCHAR" size="10" key="true">
    <DIAttributes>
      <DIAttribute name="Description" value="" />
      <DIAttribute name="ui_mapping_text"
        value="ODS_CUSTOMER.CUST_ID" />
    </DIAttributes>
  </DIElement>

  <DIElement name="CUST_CLASSF" datatype="VARCHAR" size="2">
    <DIAttributes>
      <DIAttribute name="Description" value="" />
      <DIAttribute name="ui_mapping_text"
        value="ODS_CUSTOMER.CUST_CLASSF" />
    </DIAttributes>
  </DIElement>

  ...
</DISchema>
```

Define the expression for each output field with DIExpression elements within a DIProjection block. For exported XML, additional XML syntax for the output field may be present, but only the form used in the expr attribute is required.

```
<DISelect>
<DIProjection>
  <DIExpression isString="true" expr="ODS_CUSTOMER.CUST_ID">
    <COLUMN_REFERENCE qualifier1="ODS_CUSTOMER"
      column="CUST_ID" />
  </DIExpression>
  <DIExpression isString="true"
    expr="ODS_CUSTOMER.CUST_CLASSF" />
</DIProjection>
```

Specify the input schema in the DIFrom element.

```
<DIFrom>
  <DITableSpec name="ODS_CUSTOMER" />
</DIFrom>
```

Define the SQL where clause with a DIExpression element within a DIWhere block. For exported XML, additional XML syntax for the expression may be present, but only the form used in the expr attribute is required. The format is the same as it appears in the Designer.

```
<DIWhere>
  <DIExpression isString="true"
    expr="((ODS_CUSTOMER.CUST_TIMESTAMP &gt;=
      $GV_STARTTIME) AND$#xA; (ODS_CUSTOMER.CUST_TIMESTAMP
      &lt;= $GV_ENDTIME))">
    <!-- Additional XML syntax -->
  </DIExpression>
</DIWhere>
</DISelect>
```

Apply query-specific attributes with a DIAttribute block.

```
<DIAttributes>
  <DIAttribute name="distinct_run_as_separate_process"
    value="no" />
  <DIAttribute name="group_by_run_as_separate_process"
    value="no" />
  <DIAttribute name="join_run_as_separate_process"
    value="no" />
  <DIAttribute name="order_by_run_as_separate_process"
    value="no" />
  ...
</DIAttributes>
```

End the query transform definition with the closing tag of the DIQuery element.

```
</DIQuery>
```

### Available DIAttribute names

The available DIAttribute names correspond to object parameters and properties, but do not necessarily match the wording or location used in the Designer. For a complete description of each attribute, see the “Descriptions of objects” section in the *Designer Guide*.

| DIAttribute name | Designer |
|---|---|
| distinct_run_as_separate_process | **Query Editor > Advanced > Run DISTINCT as a separate process** |
| group_by_run_as_separate_process | **Query Editor > Advanced > Run GROUP BY as a separate process** |
| join_run_as_separate_process | **Query Editor > Advanced > Run JOIN as a separate process** |
| order_by_run_as_separate_process | **Query Editor > Advanced > Run ORDER BY as a separate process** |
| run_as_separate_process | |

## 9.3.10 Parameters and variables

You can increase the flexibility and reusability of components generated by your application by using local and global variables when designing your jobs. For more information about including variables and parameters in the design of your jobs, see the “Variables and Parameters” section of the *Designer Guide*.

### Global variables

Global variables can be accessed in XML within the DIScript element. For example, to assign a constant value to a global variable, you could use the DIAssignmentStep element:

```
<DIAssignmentStep typeId="12" variable="$GV_STARTTIME">
  <DIExpression isString="true"
    expression="'2001.01.01 00:00:000'" />
</DIAssignmentStep>
```

After the global variable is defined, you can use it elsewhere in script expressions used by your job. For example, you might insert the value of the variable into a table using a SQL query:

```
<DIFunctionCallStep typeId="23">
  <DIExpression isString="true"
    expr="sql('Target_DS', 'INSERT INTO TARGET.CDC_TIME
      VALUES ({$GV_STARTTIME})')" />
</DIFunctionCallStep>
```

For more information about using global variables in your jobs, see the “Variables and Parameters, Using global variables” section of the *Designer Guide*.

### Substitution parameters

You can import one or more substitution parameter configurations to the repository with the Import_Repo_Object web service operation, as well as the Designer. In XML, substitution parameters are specified within the DISubVarStore and SVConfigurations elements.

```
<DISubVarStore typeId="103">
<SVConfigurations>
```

A default configuration can be specified with the DIAttribute element. This element is not required to import the substitution parameters.

```
<DIAttributes>
<DIAttribute name="SV_Config_Default"
  value="Configuration_1" />
</DIAttributes>
```

Specify a substitution parameter configuration with the SVConfiguration and SubVar elements. You can include more than one SVConfiguration block to specify additional substitution parameter configurations.

```
<SVConfiguration name="Configuration_1">
<SubVar name="ReportsAddressCleanse">1</SubVar>
<SubVar name="ReportsMatch">2</SubVar>
<SubVar name="USPSProviderLevel">3</SubVar>
<SubVar name="RefFilesAddressCleanse">4</SubVar>
...
</SVConfiguration>

</SVConfigurations>
</DISubVarStore>
```

After importing the substitution parameter configurations to the repository, they can be used normally in your data flows. For more information about using substitution parameters, see the “Variables and Parameters, Substitution parameters” section of the *Designer Guide*.

## 9.3.11 Basic example

This example assumes that you have a simple job named myTestJob that calls a dataflow named myTestDataflow. The dataflow contains a flat-file source (mySource), a Query transform (FormatFields), and a flat-file target (myTarget).

Additionally, this example assumes that all objects for the job have been exported to a single XML file.

### Example

All object definitions are enclosed in the DataIntegratorExport element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<DataIntegratorExport repositoryVersion="12.2.0.0000"
  productVersion="12.2.0.0">
```

The NameDate flat-file format definition is contained in the DIFlatFileDatastore element. Field names, types, and sizes are defined in the DISchema block.

```
<DIFlatFileDatastore name="NameDate" typeId="3">

<DISchema>
  <DIElement name="FirstName" datatype="VARCHAR" size="7">
  </DIElement>
  <DIElement name="LastName" datatype="VARCHAR" size="7">
  </DIElement>
  <DIElement name="DateOfBirth" datatype="VARCHAR" size="10">
  </DIElement>
</DISchema>
```

Various attributes, such as file name, directory, file type, column delimiter, and locale settings are specified in the DIAttributes block, and the flat-file format definition is closed.

```
<DIAttributes>
<DIAttribute name="abap_file_format" value="no" />
<DIAttribute name="blank_pad" value="leading" />
<DIAttribute name="cache" value="yes" />
<DIAttribute name="column_delimiter" value="," />
<DIAttribute name="column_width" value="1" />
<DIAttribute name="column_width1" value="7" />
<DIAttribute name="column_width2" value="7" />
<DIAttribute name="column_width3" value="10" />
<DIAttribute name="date_format" value="yyyy.mm.dd" />
<DIAttribute name="datetime_format"
  value="yyyy.mm.dd hh24:mi:ss" />
<DIAttribute name="file_format" value="ascii" />
<DIAttribute name="file_location" value="local" />
<DIAttribute name="file_name" value="NameDate.txt" />
<DIAttribute name="file_type" value="delimited_file" />
<DIAttribute name="locale_codepage" value="&lt;default&gt;" />
<DIAttribute name="locale_language" value="&lt;default&gt;" />
<DIAttribute name="locale_territory"
  value="&lt;default&gt;" />
<DIAttribute name="name" value="NameDate" />
<DIAttribute name="reader_capture_data_conversion_errors"
  value="no" />
<DIAttribute name="reader_capture_row_format_errors"
  value="yes" />
<DIAttribute name="reader_error_file_name" value="" />
<DIAttribute name="reader_error_file_root_dir" value="" />
<DIAttribute name="reader_log_data_conversion_warnings"
  value="yes" />
<DIAttribute name="reader_log_row_format_warnings"
  value="yes" />
<DIAttribute name="reader_maximum_warnings_to_log"
  value="-99" />
<DIAttribute name="reader_write_error_rows_to_file"
  value="no" />
<DIAttribute name="root_dir"
  value="C:\Data Services\Tutorial Files" />
<DIAttribute name="row_delimiter" value="\n" />
<DIAttribute name="skip_row_header" value="yes" />
<DIAttribute name="table_weight" value="0" />
<DIAttribute name="time_format" value="hh24:mi:ss" />
<DIAttribute name="transfer_custom" value="no" />
<DIAttribute name="use_root_dir" value="no" />
<DIAttribute name="write_bom" value="no" />
</DIAttributes>

</DIFlatFileDatastore>
```

The myTestJob job definition is contained in the DIJob element. The job contains a single step, calling the myTestDataflow dataflow. Again, attributes related to tracing, statistics, and so on are found in the DIAttributes block.

```
<DIJob name="myTestJob" typeId="2">

<DISteps>
<DICallStep typeId="1" calledObjectType="Dataflow"
  name="myTestDataflow" >
</DICallStep>
</DISteps>

<DIAttributes>
<DIAttribute name="job_checkpoint_enabled" value="no" />
<DIAttribute name="job_collect_statistics" value="no" />
<DIAttribute name="job_collect_statistics_monitor"
  value="no" />
<DIAttribute name="job_enable_assemblers" value="yes" />
<DIAttribute name="job_enable_audit" value="yes" />
<DIAttribute name="job_enable_dataquality" value="yes" />
<DIAttribute name="job_export_repo" value="no" />
<DIAttribute name="job_export_reports" value="no" />
<DIAttribute name="job_isrecoverable" value="no" />
<DIAttribute name="job_mode" value="Multi-Process" />
<DIAttribute name="job_monitor_sample_rate" value="1000" />
```

```
<DIAttribute name="job_name" value="myTestJob" />
<DIAttribute name="job_print_version" value="no" />
<DIAttribute name="job_testmode_enabled" value="no" />
<DIAttribute name="job_trace_abapquery" value="no" />
<DIAttribute name="job_trace_all" value="no" />
<DIAttribute name="job_trace_ascomm" value="no" />
<DIAttribute name="job_trace_assemblers" value="no" />
<DIAttribute name="job_trace_audit" value="no" />
<DIAttribute name="job_trace_dataflow" value="yes" />
<DIAttribute name="job_trace_idoc_file" value="no" />
<DIAttribute name="job_trace_memory_loader" value="no" />
<DIAttribute name="job_trace_memory_reader" value="no" />
<DIAttribute name="job_trace_optimized_dataflow"
  value="no" />
<DIAttribute name="job_trace_parallel_execution"
  value="no" />
<DIAttribute name="job_trace_rfc_function" value="no" />
<DIAttribute name="job_trace_row" value="no" />
<DIAttribute name="job_trace_script" value="no" />
<DIAttribute name="job_trace_session" value="yes" />
<DIAttribute name="job_trace_sql_only" value="no" />
<DIAttribute name="job_trace_sqlfunctions" value="no" />
<DIAttribute name="job_trace_sqlloaders" value="no" />
<DIAttribute name="job_trace_sqlreaders" value="no" />
<DIAttribute name="job_trace_sqltransforms" value="no" />
<DIAttribute name="job_trace_stored_procedure" value="no" />
<DIAttribute name="job_trace_table" value="no" />
<DIAttribute name="job_trace_table_reader" value="no" />
<DIAttribute name="job_trace_transform" value="no" />
<DIAttribute name="job_trace_userfunction" value="no" />
<DIAttribute name="job_trace_usertransform" value="no" />
<DIAttribute name="job_trace_workflow" value="yes" />
<DIAttribute name="job_type" value="batch" />
<DIAttribute name="job_use_statistics" value="yes" />
<DIAttribute name="locale_codepage"
  value="&lt;default&gt;" />
<DIAttribute name="locale_language"
  value="&lt;default&gt;" />
<DIAttribute name="locale_territory"
  value="&lt;default&gt;" />
</DIAttributes>

</DIJob>
```

The myTestDataflow dataflow definition is contained in the DIDataflow element.

```
<DIDataflow name="myTestDataflow" typeId="1">
```

Transforms are invoked within the DITransforms element. Because this dataflow has three transforms, source, Query, and target, there are three corresponding sections in the DITransforms element.

```
<DITransforms>
```

The file-format source definition is contained in the DIFileSource element. The name of the output schema is specified with the DIOutputView element. By default, the schema is given the same name as the file format. However, you can change it to any unique name, provided you use the same name in later transforms.

```
<DIFileSource typeId="33" formatName="NameDate"
  filename="NameDate.txt">

<DIUIOptions>
<DIAttribute name="ui_display_name" value="mySource" />
</DIUIOptions>

<DIOutputView name="NameDate" />

<DIAttributes>
<DIAttribute name="adaptable" value="no" />
<DIAttribute name="cache" value="yes" />
<DIAttribute name="connection_port" value="no" />
<DIAttribute name="file_location" value="local" />
<DIAttribute name="name" value="NameDate" />
```

```
<DIAttribute name="reader_filename_col"
  value="DI_FILENAME" />
<DIAttribute name="reader_filename_col_size" value="100" />
<DIAttribute name="reader_filename_only" value="no" />
<DIAttribute name="reader_include_filename" value="no" />
<DIAttribute name="reader_maximum_warnings_to_log"
  value="-99" />
<DIAttribute name="root_dir"
  value="C:\Data Services\Tutorial Files" />
<DIAttribute name="table_weight" value="0" />
</DIAttributes>

</DIFileSource>
```

The FormatFields Query transform definition is contained in the DIQuery element. The output schema name is specified with the DISchema element.

```
<DIQuery typeId="122" >

<DIUIOptions>
<DIAttribute name="ui_display_name" value="FormatFields" />
</DIUIOptions>

<DISchema name="FormatFields">
```

Each output field is defined with DIElement and DIAttributes elements. The ui_mapping_text attribute for each output field is required by the Designer. For the Full Name output field, the expression shows the concatenation of two input fields (FirstName and LastName). The additional encoded text is used to maintain formatting within the Designer and is optional.

```
<DIElement name="FirstName" datatype="VARCHAR" size="7">
<DIAttributes>
<DIAttribute name="Description" value="" />
<DIAttribute name="ui_mapping_text"
  value="NameDate.FirstName" />
</DIAttributes>
</DIElement>

<DIElement name="LastName" datatype="VARCHAR" size="7">
<DIAttributes>
<DIAttribute name="Description" value="" />
<DIAttribute name="ui_mapping_text"
  value="NameDate.LastName" />
</DIAttributes>
</DIElement>

<DIElement name="DateOfBirth" datatype="VARCHAR" size="10">
<DIAttributes>
<DIAttribute name="Description" value="" />
<DIAttribute name="ui_mapping_text"
  value="NameDate.DateOfBirth" />
</DIAttributes>
</DIElement>

<DIElement name="Full Name" datatype="VARCHAR" size="20">
<DIAttributes>
<DIAttribute name="Description" value="" />
<DIAttribute name="ui_mapping_text"
  value="NameDate.FirstName ||
    NameDate.LastName&#xD;&#xA;&#xD;&#xA;" />
</DIAttributes>
</DIElement>

</DISchema>
```

The SQL select projection syntax is contained in the DISelect and DIProjection elements, and varies depending on how the Query transform is configured.

```
<DISelect>
<DIProjection>
```

Each output field is defined in a DIExpression element. The expr attribute contains the actual ATL expression as displayed in the Designer. Note that each field contains an additional XML representation of the expression. This additional representation is optional, and not required for correct operation.

```
<DIExpression isString="true" expr="NameDate.FirstName">
<COLUMN_REFERENCE qualifier1="NameDate" column="FirstName" />
</DIExpression>

<DIExpression isString="true" expr="NameDate.LastName">
<COLUMN_REFERENCE qualifier1="NameDate" column="LastName" />
</DIExpression>

<DIExpression isString="true" expr="NameDate.DateOfBirth">
<COLUMN_REFERENCE qualifier1="NameDate"
  column="DateOfBirth" />
</DIExpression>

<DIExpression isString="true" expr="(NameDate.FirstName ||
  NameDate.LastName)">
<CONCAT>
<COLUMN_REFERENCE qualifier1="NameDate" column="FirstName" />
<COLUMN_REFERENCE qualifier1="NameDate" column="LastName" />
</CONCAT>
</DIExpression>

</DIProjection>
```

The input schema for the Query is defined using the DIFrom element. Attributes for the Query are specified in a DIAttributes block.

```
<DIFrom>
<DITableSpec name="NameDate" />
</DIFrom>
</DISelect>

<DIAttributes>
<DIAttribute name="distinct_run_as_separate_process"
  value="no" />
<DIAttribute name="group_by_run_as_separate_process"
  value="no" />
<DIAttribute name="join_run_as_separate_process"
  value="no" />
<DIAttribute name="order_by_run_as_separate_process"
  value="no" />
<DIAttribute name="run_as_separate_process" value="no" />
</DIAttributes>

</DIQuery>
```

The file-format target is defined within the DIFileTarget element

```
<DIFileTarget typeId="3" formatName="NameDate"
  filename="NameDate_out.txt">

<DIUIOptions>
<DIAttribute name="ui_display_name" value="myTarget" />
</DIUIOptions>
```

The input schema for the target is specified using the DIInputView element, and attributes for the target are specified using a DIAttributes block.

```
<DIInputView name="FormatFields" />

<DIAttributes>
<DIAttribute name="connection_port" value="no" />
<DIAttribute name="file_location" value="local" />
<DIAttribute name="isstreamdebugfile" value="no" />
<DIAttribute name="loader_load_choice" value="replace" />
<DIAttribute name="name" value="NameDate" />
<DIAttribute name="root_dir" value="D:\temp" />
<DIAttribute name="validate_decimal_data" value="yes" />
</DIAttributes>
```

```
</DIFileTarget>

</DITransforms>
```

Attributes for the myTestDataflow dataflow are specified in a DIAttributes block, and the file is finished by closing the DataIntegratorExport element.

```
<DIAttributes>
<DIAttribute name="Cache_type" value="pageable_cache" />
<DIAttribute name="Parallelism_degree" value="0" />
<DIAttribute name="allows_both_input_and_output"
  value="yes" />
<DIAttribute name="run_once" value="no" />
<DIAttribute name="use_dataflow_links" value="no" />
<DIAttribute name="use_datastore_links" value="yes" />
<DIAttribute name="validation_xform_exists" value="no" />
<DIAttribute name="validation_xform_stats" value="no" />
</DIAttributes>

</DIDataflow>

</DataIntegratorExport>
```

# Job launcher execution commands

The job launcher, exported as part of a job's execution commands, includes a specific command line option for server groups. You can use this option to change the job servers in a server group.

For complete information about the job launcher, see the *Management Console Guide*.

The following table lists job launcher flags and their values.

| Flag | Value |
|---|---|
| `-w` | The job launcher starts the job(s) and then waits before passing back the job status. If -w is not specified, the launcher exits immediately after starting a job. |
| `-t` | The time, in milliseconds, that the Job Server waits before checking a job's status. This is a companion argument for `-w`. |
| `-s` | Status or return code. 0 indicates successful completion, non-zero indicates an error condition.<br>Combine `-w`, `-t`, and `-s` to execute the job, wait for completion, and return the status. |
| `-C` | Name of the engine command file (path to a file which contains the Command line arguments to be sent to the engine). |
| `-v` | Prints `AL_RWJobLauncher` version number. |
| `-S` | Lists the server group and Job Servers it contains using the following syntax:<br>`"SvrGroupName;JobSvr1Name:JobSvr1Host:JobSvr1Port;JobSvr2Name:JobSvr2Host:JobSvr2Port";`<br>For example: `"SG_DEV;JS1:HPSVR1:3500;JS2:WINSVR4:3505";` |
| `-R` | The location and name of the password file. Replaces the hard-coded repository connection values for `-S`, `-N`, `-U`, `-P`. |

There are two arguments that do not use flags:

- `inet` address—The host name and port number of the Job Server. The string must be in quotes. For example:

  ```
  "inet:HPSVR1:3500"
  ```

  If you use a server group, inet addresses are automatically rewritten using the `-S` flag arguments. On execution, the first Job Server in the group checks with the others and the Job Server with the lightest load executes the job.

- `server log path`—The fully qualified path to the location of the log files. The server log path must be in quotes. The server log path argument does not appear on an exported batch job launch command file. It appears only when the software generates a file for an active job schedule and stores it in the following directory: <*DS_COMMON_DIR*>/Log/*JobServerName*/*Repository Name*/*JobInstanceName*

  You cannot manually edit server log paths.

For complete information about the job launcher, see the *Management Console Guide*.

# Legacy adapter information

## 11.1 Legacy adapter for external web services

**Caution:**
This section is provided for legacy reference only. For improved performance in new web service data flows, use the native web service datastore type.

You can add functionality to SAP BusinessObjects Data Services to invoke web services in external applications from data flows. This functionality requires configuring the software's built-in Web Services Adapter. The Web Services Adapter provides support for locating and importing metadata for a web services server as well as invoking web service operations.

The Web Services Adapter works by sending a request and waiting until it receives a reply from a web services server.

For example, you might create a web services server as a front-end to a legacy application. You could call the web services server daily from a data flow to access inventory and update an inventory data mart.

The interaction between the Web Services Adapter and an external web service has these parts:

- Creating an adapter datastore that identifies the WSDL, which describes the web services server.
- Importing metadata to extract the information form the WSDL needed to access the web service server.
- Creating a data flow that uses the imported function call to call the web services server.

**Related Topics**

- To access a web service using the Designer

### 11.1.1 Legacy adapter installation

The Web Services Adapter is part of each Job Server installation. The installer automatically configures an adapter instance in the Administrator, which is the only adapter instance that SAP BusinessObjects Data Services requires to configure a web services client. You do not need to configure adapter operations. The software automatically configures the Web Services Adapter with **Autostart** set to

FALSE so that it does not consume resources when you do not use Web services. However, you can invoke adapters set with **Autostart** disabled, when needed. You do not need to edit the adapter instance that the software provides for the Web Services Adapter.

The installer allows you to configure a Job Server to manage adapters by presenting a list of Job Servers to you during installation. To view any adapter instance in the Administrator, select **Adapter Instance > *Job Server***.

The software creates the following values for an adapter instance.

| Field name | Value automatically inserted |
|---|---|
| Adapter Instance Name | WebService |
| Access Sever host | (not required) Blank |
| Access Server port | (not required) Blank |
| Adapter Retry count | (default value) 0 |
| Adapter Retry Interval | (default value) 3000 |
| ClassPath | JAR files required in the classpath to start the Java process:<br>*LINK_DIR*/lib/acta_adapter_sdk.jar<br>*LINK_DIR*/lib/acta_broker_client.jar<br>*LINK_DIR*/lib/acta_tool.jar<br>*LINK_DIR*/ext/lib/xerces.jar<br>*LINK_DIR*/lib/acta_webservice_adapter.jar<br>*LINK_DIR*/ext/lib/qname.jar<br>*LINK_DIR*/ext/lib/axis.jar<br>*LINK_DIR*/ext/lib/commons-logging.jar<br>*LINK_DIR*/ext/lib/commons-discovery.jar<br>*LINK_DIR*/ext/lib/wsdl4j.jar<br>*LINK_DIR*/ext/lib/saaj.jar<br>*LINK_DIR*/ext/lib/jaxrpc.jar |
| AutoStart | TRUE |
| Trace Mode | FALSE (If set to TRUE, the adapter writes trace messages to the WebService_trace.txt file in the <*DS_COMMON_DIR*>/adapters/log directory) |
| Additional Java launcher options | (default values) -Xms64m -Xmx256m |

| Field name | Value automatically inserted |
|---|---|
| Adapter type name | (Read-only) Name of adapter used to create this instance |
| Adapter version | (Read-only) Version of adapter used to create this instance |
| Adapter class | (Read-only) Name that identifies the adapter entry point |

### 11.1.2 Legacy adapter configuration

To configure access to a specific web service, use the Designer. In the Designer's Datastore Editor window, specify the datastore as an adapter datastore, select the Job Server that is managing the Web Services Adapter, and select the Web Services Adapter. Data Services provides access to web services as stream-oriented function calls, which it configures when you import metadata.

When you configure an adapter datastore, in addition to the normal adapter settings, specify the URL of the web services server for a data flow to access. It must be the same URL that accepts a web service connection and returns the WSDL.

The adapter connects to the web services server using the URL to locate the definitions of published services.

#### 11.1.2.1 To access a web service

1. Create an adapter datastore:
   a. Use the Web Services Adapter instance that SAP BusinessObjects Data Services automatically creates during installation.
   b. In the datastore editor under Adapter Options, configure the following parameters:
      - **URL of the Web Service** — Enter the URL of the web services server. This URL must accept a web service connection and return the WSDL. This information is required for data flow access.
      - **XML Recursion Level** — Enter the number of passes the software should run through the XSD to resolve names. The default is 0.
      - **Keystore Path** — If the web services server uses an SSL connection, specify the location of the keystore used to establish the connection. When unsure, contact your network administrator.
      - **User Name** — Enter your user name for HTTP basic authentication.
      - **Password** — Enter your password for HTTP basic authentication.

**Note:**
To obtain user name and password information, contact your web services provider.

c. Click **OK**.

2. Import metadata:
   a. From the object library, double-click a Web Services Adapter datastore.

      The Designer calls the adapter. The adapter calls the web services server at the indicated WSDL URL and obtains a list of published services and ports.

   b. Expand the ports to see the published operations available for import.

      The list reflects the name and description of operations currently published by the configured web service.

   c. Right-click an operation and select **Import**.

      The software imports web service operations as function calls and lists them under the Web Services Adapter datastore in the object library. Each function call includes a definition for both the input and output messages required for communication with a web service operation. The adapter extracts the details about the request and reply messages and generates XML Schema files that describe the messages.

3. From the Designer, add a web service function call to a job.

   As a web services client, the software calls a web services server twice:

   - During design time to import metadata for the functions and data types that a particular web service supports.
   - During run time to call the web service and invoke this functionality.

#### 11.1.2.2 To add web service calls to a job

Once an adapter datastore is created and metadata is imported, use the following procedure to add a function call to an SAP BusinessObjects Data Services job.

1. Open the Designer.
2. Create a Web Services Adapter datastore.
3. Import operation metadata from an external web service.
4. Add a query to your job.
5. Open the query editor, right-click the target schema and select **New Function call**.

   The Function Editor opens listing the operation metadata that you imported under its datastore name.

6. Select a datastore to view the metadata that you want to add to your job.
7. Select the metadata name and click **Next**.
8. Map the input schema to the output schema.

**Note:**
If you want to nest data in the target schema, use this first query to place the schema in your job and additional queries to perform the nesting. The Function Editor does not allow complex schema configuration.

9. Click **OK**.

   The imported schema appears in the query.

10. Configure the remainder of your job by supplying input to the function call and extracting the response information obtained from the web service.

### 11.1.3 Configuring SSL with the legacy adapter

With Secure Socket Layer (SSL), the web services adapter can use secure transport over TCP/IP networks.

The overall process is:

- Generate certificates and keystores for both the server and client.
- Configure the web server.
- Configure the SAP BusinessObjects Data Services web services adapter.

#### 11.1.3.1 To generate certificates and keystores

1. Generate the server keystore.
2. Export the certificate from the server keystore to a file and get it signed by an authorized Certificate Authority.
3. Generate the client keystore.
4. Export the certificate from the client keystore to a file and get it signed by an authorized Certificate Authority.
5. Import the client's certificate into the server's keystore.
6. Import the server's certificate into client's keystore.

#### 11.1.3.2 To configure Tomcat and the legacy adapter

1. Uncomment the following entry from the `server-di.xml` file in the `TOMCAT_HOME/conf` directory.

```
<Connector port="8443"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" debug="1" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS" />
```

2. Inside the `Connector` tag, add/update the value of the `keystoreFile` and `keystorePass` parameters. The `keystoreFile` parameter should contain the *.keystore file path created when generating the keystore. The `keystorePass` parameter should contain the password used to create the keystore when generating it.
3. You might also need to modify the `wrapper.properties` file in the `LINK_DIR\ext\webserver\conf` directory. In the section `Defining the classpath`, ensure the following third-party libraries are in the path:

```
wrapper.class_path=$(ACTAHOME)\ext\lib\jnet.jar
wrapper.class_path=$(ACTAHOME)\ext\lib\jsse.jar
wrapper.class_path=$(ACTAHOME)\ext\lib\jcert.jar
```

4. After completing the configuration changes, restart the web server. If the web server starts successfully, then you should be able to access any web application supported by Tomcat via SSL.
5. In the Designer, configure the web services adapter by opening the adapter datastore, click **Advanced**, and enter the client keystore path.

### 11.1.4 Legacy adapter error messages

In addition to the error logs, the following list identifies web services client error messages and their descriptions:

- Web services client is unable to create a SOAP request to send to a server. Error = <adapter-generated exception message>

  The Web Services Adapter returns this error message if the XML message passed from SAP BusinessObjects Data Services as a Web Services Adapter function call could not be packaged into a SOAP Envelope. Processing stops before a call to a web services server is made.

  To find extended error information, see the Web Services Adapter trace log file (**Adapter Instances > *JobServerName***). To use extended diagnostics, use debug tracing in the `webadmin.log` file.

- Web services client is unable to invoke a web services server. Error = <adapter-generated exception message>

  The client returns this error message if the Web Services Adapter cannot call the web services server. It indicates that the adapter has successfully packaged the XML message passed from the software into a SOAP Envelope. However, the call to the web services server is not going through. In most cases, this will be an error in locating the service, not a case of the service refusing a request. If a web service operation is refusing a request, it will return a fault message.

To find extended error information, see the Web Services Adapter trace log file (**Adapter Instances > *JobServerName***). To use extended diagnostics, use debug tracing in the `webadmin.log` file.

- Web services client called a web services server. The server returned the following fault message: **<**server-generated error message**>**

  The client returns this error message if the web services server is called and returns a fault message indicating the call failed. The adapter has successfully packaged the XML message passed from the software into a SOAP Envelope, called the web services server, and the server received the call.

  To find extended error information, see the Web Services Adapter trace log file (**Adapter Instances > *JobServerName***) and perhaps the server itself if it maintains diagnostics. To use extended diagnostics, use debug tracing in the `webadmin.log` file.

- Web services client called a web services server and received a reply that cannot be interpreted. Error = **<** adapter-generated exception message **>**

  The client returns this error message if the SOAP Envelope returned from the web services server cannot be unpacked to extract the XML message to be returned to the job.

  To find extended error information, see the Web Services Adapter trace log file (**Adapter Instances > *JobServerName***). To use extended diagnostics, use debug tracing in the `webadmin.log` file.

# Index

## O



## R



## S



## U



## W



## X